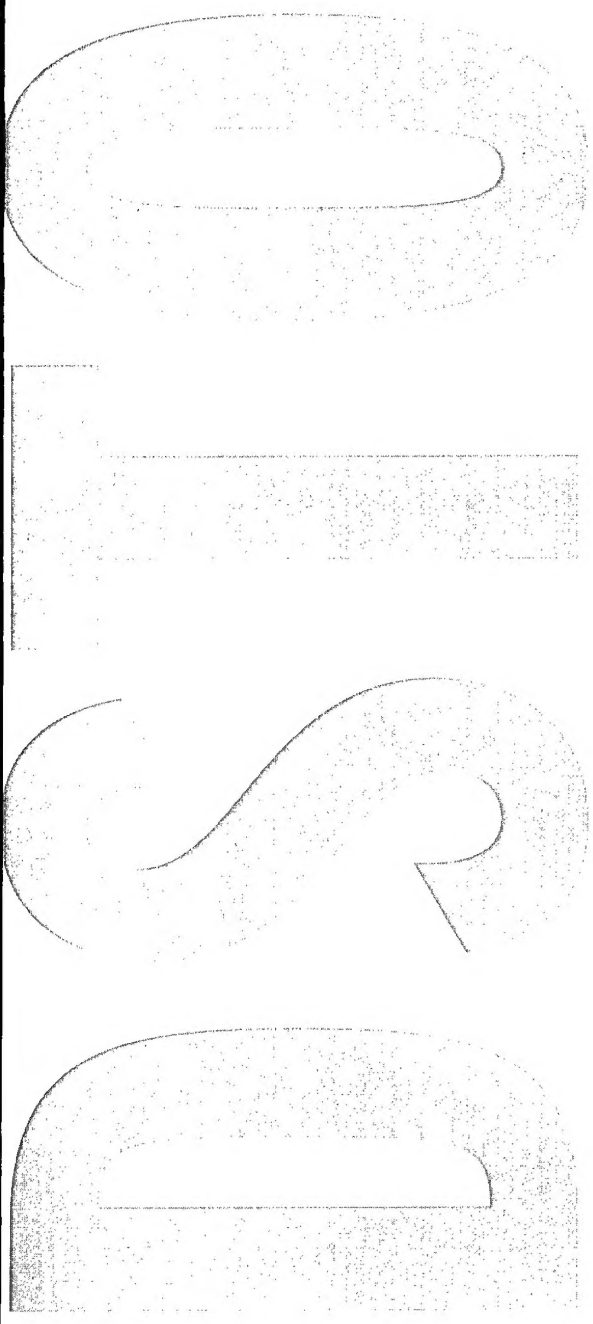
DEPARTMENT OF DEFENCE
DEFENCE SCIENCE & TECHNOLOGY ORGANISATION
DSTO



Virtual Ship Architecture Description Document Issue 1.00

Issued by John P. Best

DSTO-GD-0257

Virtual Ship Architecture Description Document Issue 1.00

Issued by
John P. Best

Maritime Operations Division
Aeronautical and Maritime Research Laboratory

DSTO-GD-0257

ABSTRACT

The Virtual Ship concept calls for simulation models of systems to be brought together to create a virtual representation of a warship, in a process analogous to the construction of a physical ship. The essential technical requirement enabling the Virtual Ship is an architecture for linking simulation models. The High Level Architecture (HLA) has been adopted and extended for this purpose. This extension is known as the Virtual Ship Architecture (VSA), and is described in this document.

RELEASE LIMITATION

Approved for public release

DEPARTMENT OF DEFENCE
DEFENCE SCIENCE & TECHNOLOGY ORGANISATION
DSTO

Published by

DSTO Aeronautical and Maritime Research Laboratory
PO Box 4331
Melbourne Victoria 3001 Australia

Telephone: (03) 9626 7000
Fax: (03) 9626 7999

AR-011-612
October 2000

APPROVED FOR PUBLIC RELEASE

Virtual Ship Architecture Description Document Issue 1.00

Executive Summary

The Virtual Ship concept calls for simulation models of systems to be brought together to create a virtual representation of a warship, in a process analogous to the construction of a physical ship. Typical components of the Virtual Ship will model the behaviour of sensor, weapon, navigation, countermeasure and command and control systems. In addition, components will model the physical behaviour of the platform, including its hydrodynamics and the effect on it of weapon detonations.

The essential technical requirement enabling the Virtual Ship is an architecture for linking simulation models. Reuse of components is a critical characteristic of the Virtual Ship and will only be achieved if simulation models are engineered in accordance with a widely accepted architecture that primarily enables the exchange and common interpretation of data, and coordination of the time advance of components. The High Level Architecture (HLA) has been adopted and extended for this purpose. This extension is known as the Virtual Ship Architecture (VSA), and is described in this document.

The central component of the Virtual Ship Architecture is the Virtual Ship Federation Object Model (VS-FOM) that documents the data that may be exchanged between the simulation models that make up the Virtual Ship. The nature of the FOM is determined, to a large extent, by the types of simulation models, or federates, that make up the Virtual Ship. Specification of these is thus a key component of the architecture.

Definition of the architecture includes specification of the coordinate system with respect to which the kinematics of the entities within the problem space are specified. Other components of the architecture, described in this document, are

1. the concept for management of a simulation execution,
2. the concept for treatment of the environment,
3. the concept for treatment of propagation effects,
4. the concept for treating the interaction between active and passive sensors.

Page deliberately left blank

Revision Record

The controlled version of this document resides in the Virtual Ship Project System Analysis and Design Folder. Any copy of this document, electronic or otherwise, should be considered uncontrolled. Amendment shall be by whole document replacement. Proposals for change should be forwarded to the issuing authority.

Issue	Date	Author	Description of Revision
1.00	20Jul00	J. Best *et al.*	First issue.

Page deliberately left blank

Authors of Issue 1.00

John Best
Anthony Cramp
Jonathan Legg
Nick Luckman
Shane Canney
Darren Sutton
Belinda Hermans
Grant Horsfall
Marcus Saunders
Amir Anvar

Page deliberately left blank

Contents

Page deliberately left blank

1. The Virtual Ship concept

The Virtual Ship concept calls for simulation models of systems to be brought together to create a virtual representation of a warship, in a process analogous to the construction of a physical ship. Typical components of the Virtual Ship will model the behaviour of sensor, weapon, navigation, countermeasure and command and control systems. In addition, components will model the physical behaviour of the platform, including its hydrodynamics and the effect on it of weapon detonations.

It is intended that the Virtual Ship be readily configured according to particular applications. It may be configured to represent in-service platforms, or concepts for future or upgraded platforms. It may be configured to represent a subset of warship functionality. Reuse of components is a critical requirement in enabling this characteristic.

A Virtual Ship capability has the potential to contribute to many processes that support platform acquisition and ownership, including capability development, acquisition, training, mission rehearsal and tactical development. A significant focus is human-in-the-loop simulation enabling investigation of the human contribution to complex system performance.

The essential technical requirement enabling the Virtual Ship is an architecture for linking simulation models. As noted above, reuse of components is critical. Reuse will only be achieved if simulation models are engineered in accordance with a widely accepted architecture that primarily enables the exchange and common interpretation of data, and coordination of the time advance of components. The High Level Architecture (HLA) has been adopted and extended for this purpose. This extension is known as the Virtual Ship Architecture (VSA), and is described in this document.

2. Structure of this document

This document provides the fundamental description of the Virtual Ship Architecture (VSA). It should be read in conjunction with *Coordinate Usage in the Virtual Ship Architecture* [1], *Execution Management in the Virtual Ship Architecture* [2], and the Virtual Ship Federation Object Model (VS-FOM).

Section 3 provides a description of the High Level Architecture. This is followed in section 4 by an overview of the Virtual Ship Architecture. The VSA is then described in detail, from a number of perspectives. Section 5 describes the federate view of the VSA which considers the typical simulation models, or components, that constitute an instance of the Virtual Ship. This view includes a description of the broad categories of data exchanged between the components.

The Virtual Ship Federation Object Model (VS-FOM) provides a detailed description of the data exchanged between Virtual Ship components. These data are structured into an object model and a description of this model provides an alternative view of the architecture. This is described in section 6.

Understanding the Virtual Ship Architecture is enhanced through exploring the relationship between the federate view and the VS-FOM. In section 7 an example federation is analysed to demonstrate how typical federates publish and subscribe object and interaction classes in order to provide data to, and obtain data from, the other federates in the federation.

There is a requirement to exert a degree of control over the flow of execution of a Virtual Ship simulation, giving rise to the concept for execution management. An overview of this concept is provided in section 8.

Modelling sensors is critical in military simulation. Key assumptions concerning the way sensors are modelled within the Virtual Ship Architecture are presented in section 9. Modelling signal propagation and the behaviour of many entities within a Virtual Ship federation is critically dependent upon the representation of the environment. The way in which the environment is treated is described in section 10.

Certain data sets have been constructed as part of the VSA and are described in sections 11 and 12.

The Virtual Ship Rules impose mandatory design requirements upon the components of the Virtual Ship, over and above those of the High Level Architecture. These are described in section 13.

Section 14 describes time management within the Virtual Ship.

Section 15 provides a description of the methodology for evolution of the Virtual Ship Architecture and highlights some areas for attention.

3. The High Level Architecture

The High Level Architecture (HLA) [3] is a framework that enables distributed simulation. It is designed to supersede both the Distributed Interactive Simulation (DIS) protocol and the Aggregate Level Simulation Protocol (ALSP). HLA has been developed by the Defense Modeling and Simulation Office (DMSO) in the United States and has been mandated for use in performing distributed simulation throughout their Department of Defense. NATO has similarly recommended that HLA be adopted as the common architecture for distributed simulation [4].

A functional representation of the HLA is shown in Figure 3a. A number of simulation models, or federates in HLA terminology, have been linked over a network to interact within a common virtual environment. The collection of all federates is the federation. Federation execution refers to the process of conducting the distributed simulation. The primary mechanism by which the federates interact is the exchange of data. This exchange is facilitated by the Run Time Infrastructure (RTI). The RTI may be viewed abstractly as a service provider to the federates. In practice it is implemented as software which runs over a computer network.

Figure 3a shows the different types of federate that may be linked within the HLA. They range from time and event stepped federates, through real time federates typical of training simulations, to passive federates such as data loggers and stealth viewers. Interfaces to real systems may be included within the federation.

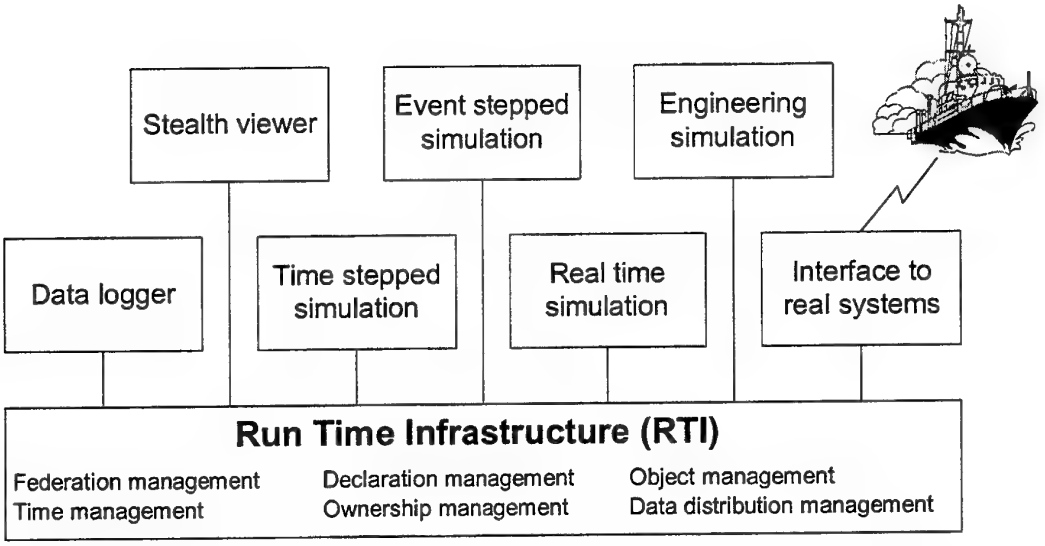


Figure 3a: Functional view of the High Level Architecture[1]. Different types of federates are linked via the Run Time Infrastructure. The six service sets facilitate interaction of these components with the object management services enabling data exchange and time management services enabling coordination of time advances.

The HLA is described formally by three components. These are the HLA Rules, the HLA Interface Specification and the HLA Object Model Template (OMT). The HLA

[1] Adapted from http://hla.dmso.mil/general/annotate/sld016.html

Rules describe mandatory characteristics of federates and federations required to facilitate reuse. The HLA Interface Specification describes the services provided by the Run Time Infrastructure and the interface to it. The Object Model Template provides a formalism for describing the data exchanged between federates.

Within the HLA the data exchanged between federates is categorised according to whether it describes some persistent entity or whether it effectively constitutes a message. The notion of an object is introduced to describe a persistent entity and those items of data that describe the characteristics of this entity and are exchanged between federates are known as attributes of the object. Information that is exchanged as a message is known as an interaction. The content of the message is in the form of a set of parameters.

Within the Object Model Template is a collection of tables for describing these data. The Object Class Structure Table details the object class names and arranges them in a hierarchy. The Attribute Table lists the attributes of the object classes. An inheritance relationship exists between object classes. Subclasses inherit all attributes of their superclasses. Similarly, the Interaction Class Structure Table details the interaction class names in a hierarchical structure. The Parameter Table lists the parameters of the interaction classes.

The capabilities of an individual federate to model certain entities, to provide information concerning them to other federates, to receive information about entities represented in other federates, and to send and receive interactions are published in the Simulation Object Model (SOM). This is constructed in accordance with the Object Model Template. The Federation Object Model (FOM) documents all objects represented in the federation, all interactions and their attributes and parameters. It is essentially a superset of the SOM's.

The federates exchange information via the RTI through the processes of publication and subscription. Federates that own object attributes publish updates of their values. Federates that wish to receive attribute updates subscribe to these attributes. Hence, when a federate publishes a new value of an attribute this is passed by the RTI to all other federates that have subscribed to this attribute. Federates also publish and subscribe interactions.

The full collection of services provided by the RTI are grouped into six management areas as follows:

1. Federation management,
2. Declaration management,
3. Object management,
4. Ownership management,
5. Time management,
6. Data distribution management.

The federation management services provide the means to create and destroy federation executions, synchronise the federation, save the state of a federation and restore a saved state.

The declaration management services provide the means by which federates advise the RTI as to their interests in providing and receiving certain data items. The term publication refers to a federate advising the RTI of its capability to provide updates of object attribute values or send interactions. The term subscription refers to a federate advising the RTI of its desire to receive updates of object attribute values or interactions.

The object management services provide the means by which federates send and receive attribute updates, send and receive interactions and alter the transportation and ordering properties of these updates.

The ownership management services provide the means by which ownership of attributes may be transferred amongst the federates. A federate that owns an attribute has responsibility for providing updates of its value. The ownership management services provide the means by which this responsibility is vested in the most capable federate. A consequence is that a single physical object represented within a federate, such as a ship, may have ownership of its attributes distributed across a number of federates.

The time management services provide the means by which federates advance their time, and particularly the means by which they are coordinated across the federation. These services support both DIS-like and ALSP-type time management approaches. In DIS there is no common time and each of the federates advances its time according to a local clock. In ALSP-type federations the federates must co-ordinate their time advances and particularly preserve causality through delivery of messages in time stamp order.

The data distribution management services provide the means by which routing spaces may be created in order to more efficiently transport attribute updates and interactions over the network. Routing spaces may typically be defined as geographic regions of interest and facilitate data filtering.

The federate developer makes use of the RTI through the application programmers interface (API). Currently there is an API implemented in C++, Ada 95, Corba IDL and Java. There are two parts of the interface to the RTI. These are the RTI Ambassador and the Federate Ambassador. The RTI Ambassador is a library that is linked into each federate and facilitates communication from the federate to the RTI. The Federate Ambassador facilitates communication from the RTI to the federate. Although the interface to the Federate Ambassador is mandated, the simulation developer must construct the internal details in order to implement the appropriate response to messages from the RTI. As an example, when a federate is required to issue an attribute update this is achieved through invocation of an RTI Ambassador method. When this attribute update is received at another federate, the local RTI component (LRC) at this federate makes a call-back to the appropriate Federate Ambassador method.

4. The Virtual Ship Architecture - An overview

The HLA confers a great deal of flexibility on the federation developer. This fundamentally stems from the freedom to construct the Federation Object Model according to the information capabilities and needs of the specific federates. This alludes to one approach to constructing the FOM. The data provision capabilities and needs of each federate are analysed and used to construct the FOM. A potential deficiency with this approach is that federates not initially used to construct the FOM may not be able to participate in this federation owing to the specific nature of the FOM.

A principal component of the Virtual Ship concept is reuse of federates across applications and use of federates from a multitude of sources. In essence, all potential users and contributors form a user group, with a shared interest in simulation interoperability in the maritime domain. An additional interest of the group is the capability to link models that represent the systems that compose warfare entities in the maritime domain.

An approach to this objective is to impose constraints additional to the HLA on the design of federates that are brought together to form instances of the Virtual Ship. For example, the imposition of a common Federation Object Model ensures the minimum condition that federates brought into the Virtual Ship can exchange data and interpret it in a common manner. This collection of additional constraints is considered to form the Virtual Ship Architecture (VSA).

It is not intended that the VSA provide a sufficient condition that federates designed in accordance with it will interoperate in a valid way. It is intended to provide a necessary condition and sufficiency will need to be proven through the design methodology applied in construction of individual instances.

The central component of the Virtual Ship Architecture is the Virtual Ship Federation Object Model (VS-FOM). The construction of the FOM incorporates construction of a lexicon of terms that facilitates common semantic interpretation of data. The nature of the FOM is determined, to a large extent, by the types of federates that make up the Virtual Ship. Specification of the types of federates that make up the Virtual Ship is thus a key component of the architecture.

It is required to specify the kinematics of the entities within the problem space with respect to common coordinate systems. Definition of these is a key component of the architecture.

Simulation of the physical environment often requires access to data sets that describe, for example, the characteristics of the environment in a form appropriate to modelling the performance of sensors. This alludes to a number of requirements. First is the requirement to adopt data standards, not limited to environmental data. Second is adoption of an approach to modelling signal propagation. In any warfare domain, and particularly the maritime domain, the environment is perceived through sensors, and their operation is based upon the physics of signal propagation. Third is an approach to modelling the interaction between active and passive sensors.

Although the HLA provides mechanisms for coordination of the time advance of federates, a need has been recognised to provide an additional level of control over the flow of federation execution. For example, in a real time application, it is desirable to ensure that all federates required for the simulation have joined the federation before the simulation proper begins. Such functionality needs to be supplied by an execution manager operating according to a well-defined concept for execution management.

In summary, key items that constitute the Virtual Ship Architecture 1.0 are

1. a description of the types of federates that make up the Virtual Ship,
2. the Virtual Ship Federation Object Model,
3. a standard for coordinate usage,
4. the concept for execution management,
5. the concept for treatment of the environment,
6. the concept for treatment of propagation effects,
7. the concept for treating the interaction between active and passive sensors.

The original statement of requirement for VSA 1.0 is attached for information as Annex A.

The use of coordinates in the VSA is described in the separate document *Coordinate Usage in the Virtual Ship Architecture* [1]. An overview of the concept for execution management is given in section 8 and described in detail in the document *Execution Management in the Virtual Ship Architecture* [2]. The treatment of propagation and the interaction between active and passive sensors are described in section 9. Treatment of the environment is described in section 10.

Understanding of the VSA is facilitated by description of the various views of the Virtual Ship that the architecture provides. In the next section the physical, or federate, view is described. Typical federates that make up the virtual ship are described, with particular emphasis on their capabilities to provide, and their requirements to receive, data. Typical ways in which input data is processed and output data generated are considered.

Following this section is a description of the view provided by the VS-FOM. The Object Class Structure table defines typical objects that are considered to exist within the Virtual Ship and these are described. Consideration of how typical federates' data provision capabilities and needs are accommodated through publication and subscription provides the link between the federate and VS-FOM views of the architecture, and this is described in section 7.

5. Physical view of the Virtual Ship Architecture

5.1 Introduction

The fundamental physical components of the Virtual Ship are HLA federates. The components may model entities at various levels of aggregation. Typical components will represent the systems that compose a warship, including sensor, weapon, countermeasure, navigation and command and control systems. Other components may represent complete warfare entities, or may represent sub-components of systems.

This section describes these typical federates in broad terms. A classification of the data input to, and output from, these federates is also introduced and the typical operation of the federates is described.

It needs to be borne in mind that the intention of this section is to enhance understanding of the typical components of a Virtual Ship federation. The categorisation of federates is not intended to be definitive, nor is the categorisation of the data exchanged between federates. Indeed, as will become apparent, the classification of a federate may not be decisive.

5.2 Typical federates that make up the Virtual Ship

The typical components are illustrated in Figure 5.2a. The components interact via the HLA Run Time Infrastructure (RTI), with data exchange the principal component of this interaction. Data exchanged between federates is conceptually described as an object attribute update or an interaction. An object attribute update is an item of data that describes some property of an entity that persists in time. An interaction is an item of data that is transient in nature.

The typical federates that compose the Virtual Ship are as follows.

5.2.1 Navigation system federate

A navigation system federate models the behaviour of a navigation system.

5.2.2 Countermeasure system federate

A countermeasure system federate models the behaviour of a countermeasure system, including active RF countermeasures, active acoustic countermeasures and chaff launching systems. Any entity deployed from a countermeasure system, such as a cloud of chaff, may be modelled within such a federate or, in the case of complex entities, such as the Nulka hovering decoy, may be modelled within a federate classed as representing external entities (see section 5.2.8).

5.2.3 Weapon system federate

A weapon system federate models the behaviour of a weapon system. Examples include missile systems and guns. The weapons actually deployed from a weapon

system may be modelled within such a federate or, in the case of complex weapons such as missiles and torpedoes, may be modelled within federates classed as representing external entities (see section 5.2.8).

5.2.4 Command and control system federate

A command and control system federate models the behaviour of command and control system components, including data fusion, threat evaluation, weapon assignment and fire control. A command and control system federate may represent the complete functionality of a combat system, or a subset of it.

5.2.5 Sensor system federate

A sensor system federate models the behaviour of a sensor system, including sonar, radar, electronic support sensors and infrared sensors.

5.2.6 Motion federate

A motion federate models the movement of a platform. In doing so, such a federate may model components of the propulsion system.

5.2.7 Damage federate

A damage federate models the effect of weapon impact and detonation upon a platform and its systems.

5.2.8 External entity federate

An external entity federate models an entity external to the Virtual Ship. This might include friendly, enemy or neutral platforms in the air, sea or land domains, or friendly or enemy weapons/countermeasures.

5.2.9 Execution manager federate

An execution manager federate provides a fundamental level of control over the flow of federation execution.

5.2.10 Utility federate

Utility federates provide functionality such as stealth viewing and data logging. They support comprehension of the scenario and the events occurring within it. They may provide rigorous diagnostic and analysis functionality.

The sensor, weapon, countermeasure, navigation, command and control, motion and damage federates provide a representation of a warship, referred to as the Virtual Ship. Typically there are multiple systems that compose a warship, but only one representation of the motion and damage are required. There may be multiple instances of the Virtual Ship, which interact with multiple external entities. Multiple utility federates may be used and there is a single execution manager. These characteristics are graphically illustrated in Figure 5.2a.

Figure 5.2a: A graphical representation of the typical federates that participate in a Virtual Ship federation execution. The components interact via the HLA Run Time Infrastructure.

5.3 Generic classification of the data exchanged between federates

A description of the typical behaviour of the federates that compose the Virtual Ship is facilitated through consideration of the data exchanged between them. These data are grouped into broad classifications and described in the following.

5.3.1 Composite entity state data

Within the Virtual Ship Architecture, those entities that exist in their own right are known as composite entities (see section 6.1.1). Examples include ships, submarines, aircraft and missiles. The state of such an entity is defined by a finite data set, known as composite entity state data. These data include items that specify the kinematics of the entity and those that uniquely identify it.

5.3.2 Component system state data

A component system is considered to be part of a composite entity and is characterised by a data set known as the component system state data. Part of the component system state data describes the location of the system with respect to the composite entity of which it is part. This composite entity is referred to as the parent entity.

5.3.3 Sensor task data

The operation of a sensor in a particular mode is typically described as a task. For example, a radar may perform a search task, as opposed to a tracking task. Some sensors, such as phased array radars, may simultaneously perform multiple tasks. The sensor task data characterises a single sensor task.

5.3.4 Track data

Sensor systems generate a representation of the motion of entities in the form of a track. This is typically a time series of position or bearing measurements, along with additional items of information indicative of the errors in these measurements and the potential identity of the entity being tracked. These data are referred to as track data.

5.3.5 System control messages

System control messages are used to exert control over a component system. For example, a sensor system may be controlled via a command and control system. This is achieved through the command and control system sending system control messages to the sensor system.

5.3.6 Weapon/countermeasure launch messages

Weapon/countermeasure launch messages instruct relevant federates to simulate the launch of a weapon/countermeasure. For example, a command and control system may send such a message to a weapon system federate.

5.3.7 Weapon impact/detonation notification messages

Federates are advised of the occurrence of weapon impact/detonation through receipt of weapon impact/detonation notification messages. These messages are issued by federates that model weapons.

5.3.8 System damage notification messages

A system damage notification message advises that a system has sustained a measure of damage due to a weapon impact/detonation. These messages are typically sent from a damage federate to system federates.

5.3.9 Execution management messages

The Virtual Ship Execution Manager controls the flow of federation execution through issuing execution management messages (see section 8).

5.3.10 Execution management error messages

A federate may not be able to respond to an instruction issued by the Virtual Ship Execution Manager. In such a circumstance the federate will issue an execution management error message.

5.4 Example behaviour of typical federates

The following sections describe the behaviour of the typical federates that make up the Virtual Ship. The primary emphasis is upon the data flows in and out of the federates, described in terms of the above categories. In addition, the notation indicates whether the flow of data is via the mechanism of the HLA attribute update or interaction. Solid arrows illustrate data items exchanged as attribute updates and dashed lines indicate data exchanged as interactions.

5.4.1 Sensor system federate

Understanding the nature of a sensor system federate is facilitated through consideration of two cases. In the first case, the sensor detects an entity based on characteristics of the total, or composite, entity. An example of such a sensor is a radar. In the second case the sensor detects the component systems that make up a composite entity. An example of such a sensor is an electronic support sensor (ESM) that detects the electromagnetic emissions of radars and communications equipment.

A case 1 sensor system federate is illustrated in Figure 5.4.1a and a case 2 sensor system federate is illustrated in Figure 5.4.1b.

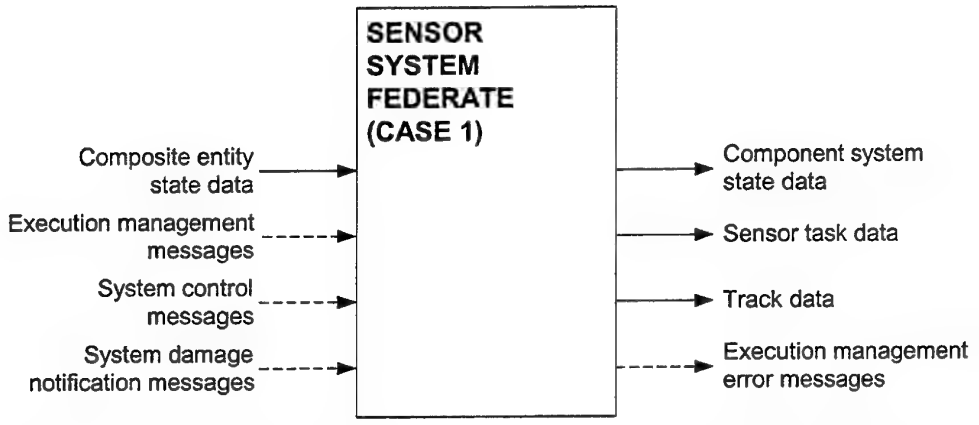


Figure 5.4.1a: The data input to, and output from, a typical sensor system federate (case 1).

5.4.1.1 Typical behaviour of a sensor system federate (case 1)

The principal role of a sensor system federate is to generate track data that describe the entities detected by it. Some fundamental information is required to fulfil this role. Propagation of signals between the sensor and entities requires information concerning the location of the sensor and the entities. The location of the sensor is derived from the location of the entity to which it is attached and the relative location of the sensor with respect to this parent entity. The location of the parent entity is derived from composite entity state data.

In the case considered here, the entity that is sensed is a composite entity. Its kinematic attributes are obtained though the composite entity state data. These data additionally identify the entity, from which information the sensor system federate computes its response. For example, if the sensor system federate represents a radar then it will utilise information concerning the identity of the sensed entity in order to associate an appropriate value for the radar cross section and therefore compute the strength of the scattered signal.

The sensor system may be controlled through receipt of system control messages. These typically cause the sensor to alter its behaviour, which may be reflected through changes in the component system state data and sensor task data. The output of these data also facilitates detection of the sensor, if it is an active sensor, by the mechanism described in section 5.4.1.2.

The sensor system federate may receive system damage notification messages, which indicate that damage may have been sustained. If the federate is so designed, it will reflect this through degraded performance.

The federate is subject to execution management and therefore receives execution management messages and responds in accordance with the concept for execution management (see section 8). In the event of an inability to respond to such direction, the sensor federate issues execution management error messages.

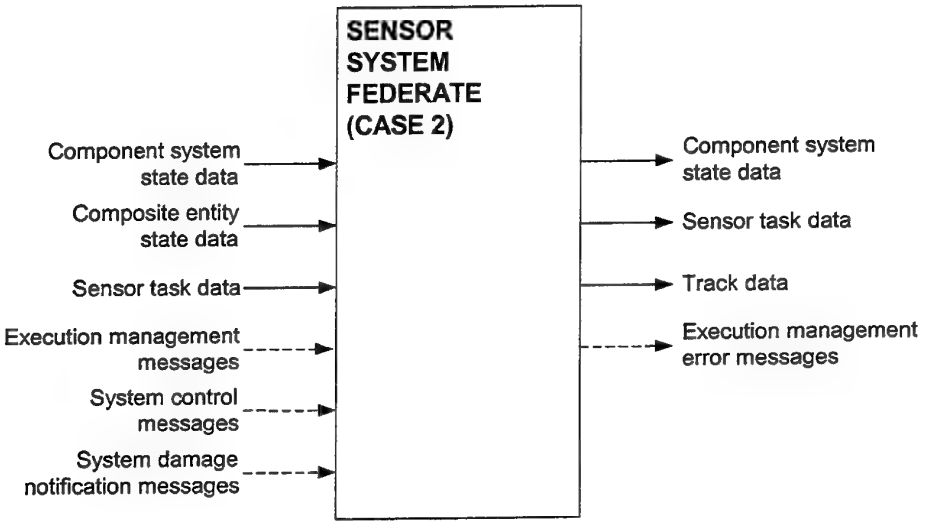


Figure 5.4.1.b: The data input to, and output from, a typical sensor system federate (case 2).

5.4.1.2 Typical behaviour of a sensor system federate (case 2)

The principal role of a sensor system federate is to generate track data that describe the entities detected by it. Some fundamental information is required to fulfil this role. Propagation of signals between the sensor and entities requires information concerning the location of the sensor and the entities. The location of the sensor is derived from the location of the entity to which it is attached and the relative location of the sensor with respect to this parent entity. The location of the parent entity is derived from composite entity state data.

In the case considered here, an emitting entity is sensed. An example is detection of a radar by an electronic support sensor. Composite entity state data provide the kinematics of the radar's parent entity, from which the location of the radar is derived. Component system state data permit identification of the radar system and sensor task data permit determination of whether signals emitted by the radar are directed towards the electronic support sensor.

The sensor system may be controlled through receipt of system control messages. These typically cause the sensor to alter its behaviour, which may be reflected through changes in the component system state data and sensor task data.

The sensor system federate may receive system damage notification messages, which indicate that damage may have been sustained. If the federate is so designed, it will reflect this through degraded performance.

The federate is subject to execution management and therefore receives execution management messages and responds in accordance with the concept for execution management (see section 8). In the event of an inability to respond to such direction, the sensor federate issues execution management error messages.

5.4.1.3 Other data required

The sensor system federate requires data describing those aspects of the environment that are relevant to its performance. Also required are data describing the scattering and/or emitting properties of the entities that may be sensed by the federate. The

system state data reflect, in part, the location of the sensor system with respect to its parent entity. These data are required in order to initialise the sensor system federate.

5.4.2 Weapon system federate

Understanding the nature of a weapon system federate is aided through consideration of two cases. In case 1, the weapon system federate models the system that causes weapons to be launched and the weapons themselves. Such an implementation is likely when the weapons themselves are simple to model and an example is a gun. In case 2, the weapon system federate models only the system that causes weapons to be launched. The weapon itself is modelled within a separate federate; typically an external entity federate (see section 5.4.8). An example of such a system is a missile system.

A case 1 weapon system federate is illustrated in Figure 5.4.2a and a case 2 weapon system federate is illustrated in Figure 5.4.2b.

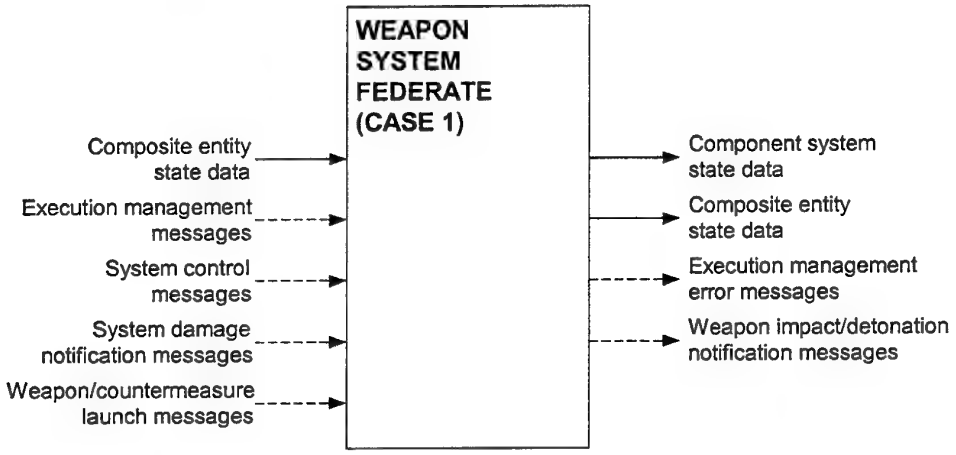


Figure 5.4.2a: The data input to, and output from, a weapon system federate (case 1).

5.4.2.1 Typical behaviour of a simple weapon system federate (case 1)

A weapon system prepares weapons for deployment and causes their deployment upon command. Such a command constitutes a weapon/countermeasure launch message. The weapon system federate models the action required to prepare a weapon for deployment, the deployment and the trajectory of the weapon until such time as it impacts upon, or detonates in the vicinity of, a target, or it ceases to pose a threat to any other entity in the battle space. A weapon impact/detonation notification message is issued when such an event is deemed by this federate to have occurred.

Modelling the interaction between the weapons deployed and potential targets requires information concerning the latter, and this is obtained through receipt of composite entity state data. In addition, receipt of these data provides the federate with data concerning the composite entity of which this component system is part.

Given that this federate models the actual weapon deployed by the system it provides composite entity state data to the federation. The weapon system may

change its behaviour in response to system control messages. Its current state is reflected through provision of component system state data to the federation.

The federate will receive execution management messages and system damage notification messages and respond as described for the sensor system federate. Execution management error messages may be issued.

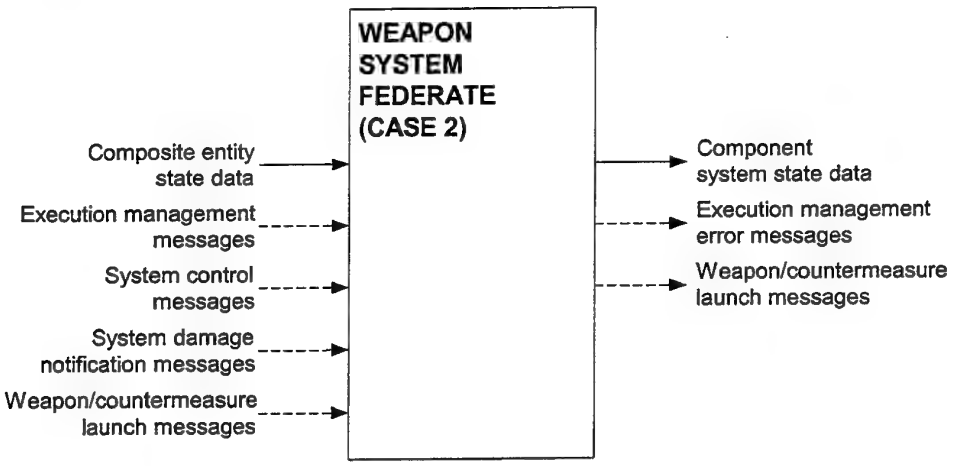


Figure 5.4.2b: The data input to, and output from, a weapon system federate (case 2).

5.4.2.2 Typical behaviour of a weapon system federate (case 2)

A weapon system prepares weapons for deployment and causes their deployment upon command. Such a command constitutes a weapon/countermeasure launch message. The weapon system federate models the action required to prepare a weapon for deployment and the deployment. The trajectory of the weapon and subsequent events are modelled within another federate, typically an external entity federate. When this federate determines that a weapon has been deployed it communicates this event to the federate modelling the weapon through a weapon/countermeasure launch message. At the simplest level of modelling the weapon deployment process, such a federate may merely relay the initial weapon/countermeasure launch message with an appropriate delay.

This federate requires composite entity state data that describe the composite entity of which this component system is part.

The weapon system may change its behaviour in response to system control messages. Its current state is reflected through provision of component system state data to the federation.

The federate will receive execution management messages and system damage notification messages and respond as described for the sensor system federate. Execution management error messages may be issued.

5.4.2.3 Other data required

The weapon system federate requires data describing those aspects of the environment that are relevant to its physical performance. The system state data

reflect, in part, the location of the weapon system with respect to its parent entity. These data are required in order to initialise the weapon system federate.

5.4.3 Countermeasure system federate

Understanding the nature of a countermeasure system federate is aided through consideration of two cases. In case 1 the countermeasure system federate models the system that causes countermeasures to be launched and the countermeasures themselves. Such an implementation is likely when the deployed countermeasure is simple and an example is chaff. In case 2 the countermeasure system federate models only the system that causes countermeasures to be launched. The countermeasure itself is modelled within a separate federate; typically an external entity federate (see 5.4.8). An example of such a system is the Nulka hovering decoy.

A case 1 countermeasure system federate is illustrated in Figure 5.4.3a and a case 2 countermeasure system federate is illustrated in Figure 5.4.3b.

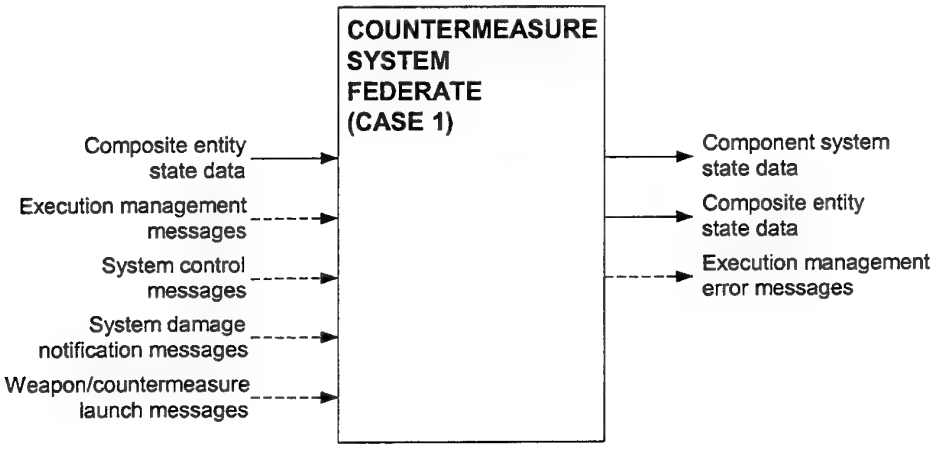


Figure 5.4.3a: The data input to, and output from, a countermeasure system federate (case 1).

5.4.3.1 Typical behaviour of a countermeasure system federate (case 1)

A countermeasure system prepares countermeasures for deployment and causes their deployment upon command. Such a command constitutes a weapon/countermeasure launch message. The countermeasure system federate models the action required to prepare a countermeasure for deployment, the deployment, the trajectory and subsequent behaviour of the countermeasure.

Modelling the interaction between the countermeasures deployed and potential targets requires information concerning the latter, and this is obtained through receipt of composite entity state data. In addition, receipt of this data provides the federate with data concerning the composite entity of which this component system is part.

Given that this federate models the actual countermeasure deployed by the system it provides composite entity state data to the federation. The countermeasure system may change its behaviour in response to system control messages. Its current state is reflected through provision of component system state data to the federation.

The federate will receive execution management messages and system damage notification messages and respond as described for the sensor system federate. Execution management error messages may be issued.

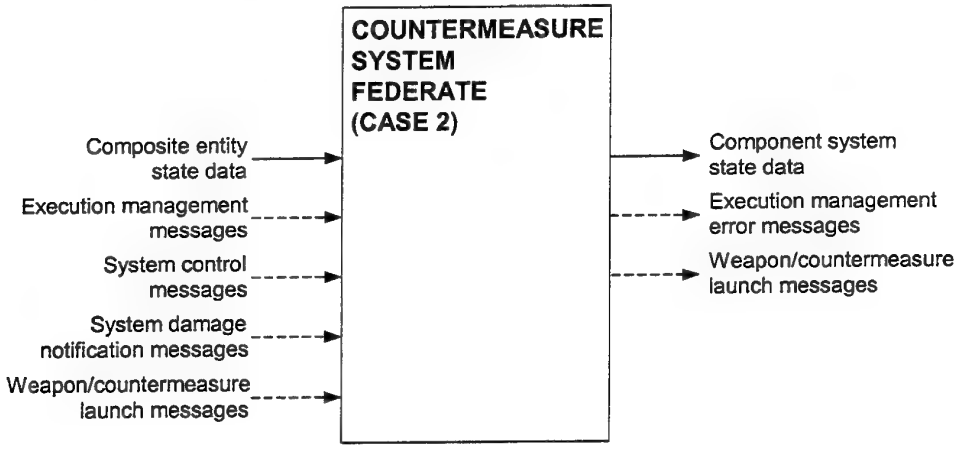


Figure 5.4.3b: The data input to, and output from, a countermeasure system federate (case 2).

5.4.3.2 Typical behaviour of a countermeasure system federate (case 2)

A countermeasure system prepares countermeasures for deployment and causes their deployment upon command. Such a command constitutes a weapon/countermeasure launch message. The countermeasure system federate models the action required to prepare a countermeasure for deployment and the actual deployment. The trajectory of the countermeasure and subsequent events are modelled within another federate, typically an external entity federate. When this federate determines that a countermeasure has been deployed it communicates this event to the federate modelling the countermeasure through a weapon/countermeasure launch message. At the simplest level of modelling the countermeasure deployment process, such a federate may merely relay the initial weapon/countermeasure launch message with an appropriate delay.

This federate requires composite entity state data that describe the composite entity of which this component system is part.

The countermeasure system may change its behaviour in response to system control messages. Its current state is reflected through provision of component system state data to the federation.

The federate will receive execution management messages and system damage notification messages and respond as described for the sensor system federate. Execution management error messages may be issued.

5.4.3.3 Other data required

The countermeasure system federate requires data describing those aspects of the environment that are relevant to its physical performance. The system state data reflect, in part, the location of the countermeasure system with respect to its parent entity. These data are required in order to initialise the countermeasure system federate.

5.4.4 Navigation system federate

A typical navigation system federate is illustrated in Figure 5.4.4a.

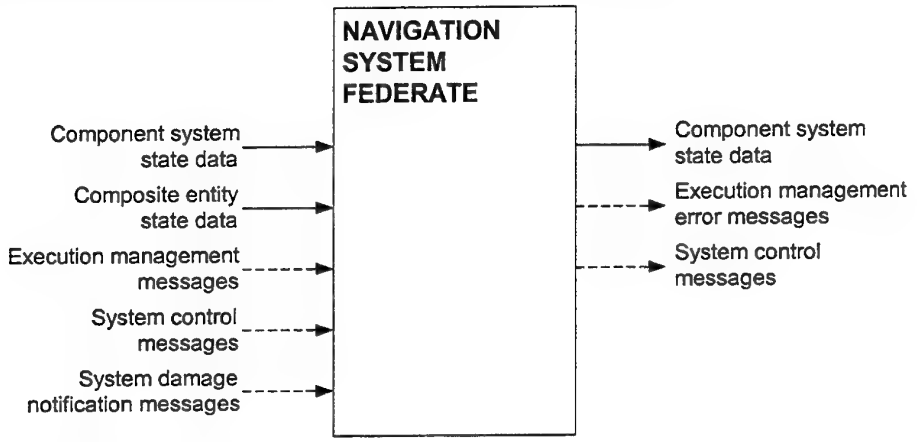


Figure 5.4.4a: The data input to, and output from, a typical navigation system federate.

5.4.4.1 Typical behaviour

The role of the navigation system federate is to provide a means for navigating a Virtual Ship. This role will be primarily executed through the issue of system control messages to facilitate changes in the course and speed of the parent entity. These will typically be received and responded to by a motion federate. As a result, the motion federate may change its behaviour which will be reflected in changes to its component system state data. The navigation system federate will receive these data in order to facilitate monitoring of the parent entity's propulsion and motion characteristics. The location of the parent entity is determined through receipt of composite entity state data.

The navigation system federate will itself be subject to control, through receipt of system control messages. These may originate from a command and control system federate. The provision by the navigation system of component system state data will facilitate such an exercise of control.

The federate will receive execution management messages and system damage notification messages and respond as described for the sensor federate. Execution management error messages may be issued.

5.4.4.2 Other data required

The navigation system federate may utilise navigation data including charts and weather information.

5.4.5 Motion federate

A typical motion federate is illustrated in Figure 5.4.5a.

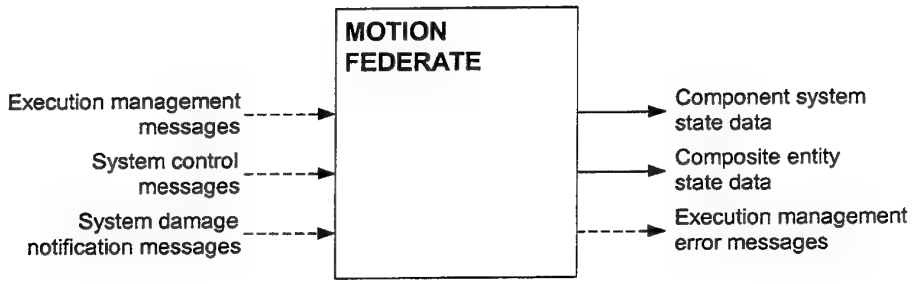


Figure 5.4.5a: The data input to, and output from, a typical motion federate.

5.4.5.1 Typical behaviour

The motion federate computes the motion of a composite entity.

The motion federate receives system control messages, typically from the navigation system federate, that indicate an intention to change the course and speed of the entity whose motion is modelled by this federate. The motion federate attempts to satisfy the request and determines the time history of kinematic attributes as the request is satisfied. These data are output for use by other federates in the form of composite entity state data. In addition, the motion federate may model aspects of the propulsion system, which is described by the output of component system state data.

The federate will receive execution management messages and system damage notification messages and respond as described for the sensor federate. Execution management error messages may be issued.

5.4.5.2 Other data required

The motion federate requires data describing those aspects of the environment that affect the motion of the entity being modelled. These data include sea and weather conditions. Also required are data describing the physical construction of the entity. In the case of a warship, these will include the hull form, details of the propulsion system, the distribution of mass within the hull and the above water surface area of the warship.

5.4.6 Damage federate

A typical damage federate is illustrated in Figure 5.4.6a.

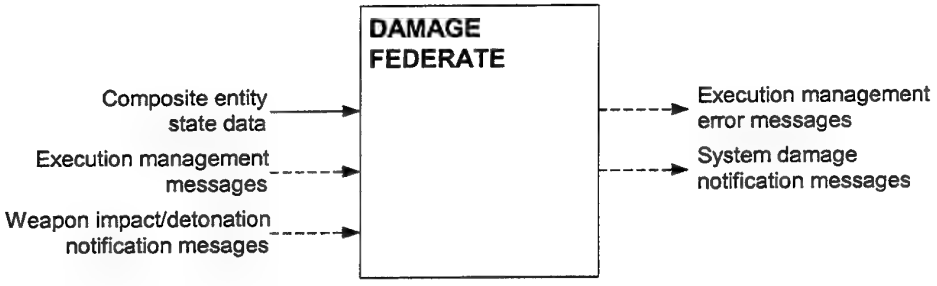


Figure 5.4.6a: The data input to, and output from, a typical damage federate.

5.4.6.1 *Typical behaviour*

The damage federate computes the damage caused to a platform and its systems by weapon impact/detonation and provides system damage notification messages to other federates. These computations and notifications are generated in response to the receipt of weapon impact/detonation notification messages.

In order to perform damage computation this federate requires composite entity state data. These data particularly permit identification of the platform.

The federate will receive execution management messages and respond in accordance with the concept for execution management. Execution management error messages may be issued.

5.4.6.2 *Other data required*

The damage federate requires data describing the physical construction of the entity whose damage is being computed. In the case of a warship, this will include the hull form, compartment layout, and the distribution and connectivity of systems.

5.4.7 Command and control system federate

A typical command and control system federate is illustrated in Figure 5.4.7a.

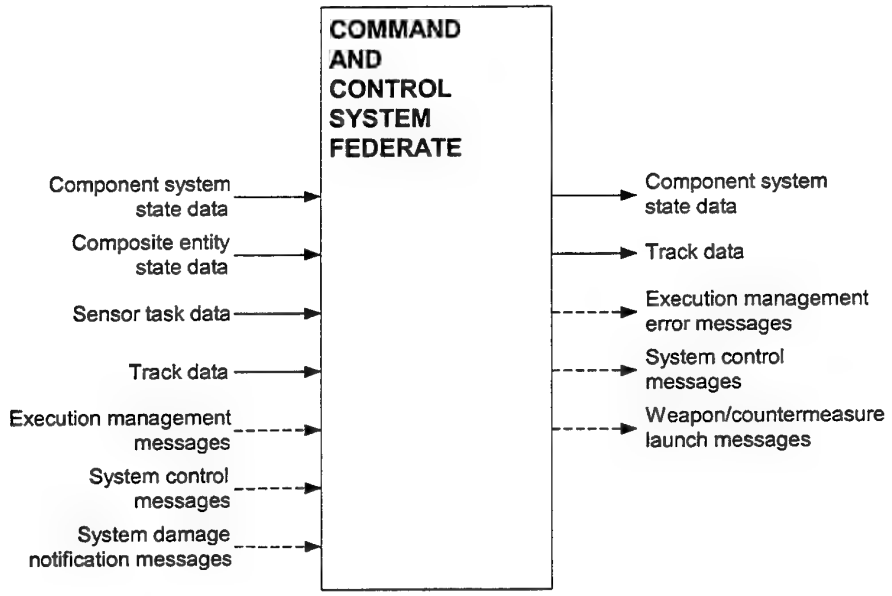


Figure 5.4.7a: The data input to, and output from, a typical command and control system federate.

5.4.7.1 *Typical behaviour*

The principal role of the command and control system federate is to provide the means by which a human operator exercises control over the many systems that compose a complex entity, such as a warship. To give effect to operator intentions, the command and control system federate issues system control messages and weapon/countermeasure launch messages.

The exercise of command and control is facilitated by the presentation of information to the operator. This information may consist of component system state data, sensor task data and track data. Track data may be fused to generate a common tactical picture, which may be presented to the operator or output from the federate in the form of fused track data. Manipulation of these data items requires information concerning the parent composite entity over which this system exercises command and control. Composite entity state data are therefore required by this federate.

A command and control system federate may provide only a subset of command and control functionality. It may therefore be itself subject to control through receipt of system control messages. To facilitate this control, component system state data describing the command and control system federate are provided to other federates.

The federate will receive execution management messages and system damage notification messages and respond as described for the sensor federate. Execution management error messages may be issued.

5.4.7.2 Other data required

Nil

5.4.8 External entity federate

A typical external entity federate is illustrated in Figure 5.4.8a.

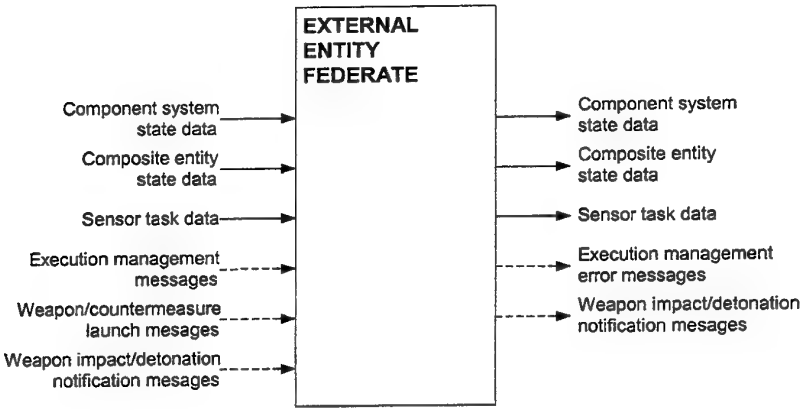


Figure 5.4.8a: The data input to, and output from, an external entity federate.

5.4.8.1 Typical behaviour

An external entity federate represents a complete composite entity, such as a ship, aircraft, missile or countermeasure. It provides composite entity state data to other federates. In addition, if the entity represented operates sensors then component system state data and sensor task data are provided. The external entity federate may be considered to provide a logical grouping of all previous federates (see section 5.5).

The federate requires composite entity state data describing other entities in the environment, in addition to component system state data and sensor task data describing the operation of sensors in the environment. These data are used by the external entity federate to compute its interaction with other entities. The performance of sensor, weapon, countermeasure, navigation and command and

control systems that make up the external entity make use of these data as described for the previous federates.

An external entity federate that represents a weapon or countermeasure will receive weapon/countermeasure launch messages and create object instances to represent weapon or countermeasure entities. In the event that a weapon is modelled, the federate will determine if the weapon impacts upon, or detonates in the vicinity of, some other entity and will output a weapon impact/detonation notification message to advise other federates of this occurrence.

If another federate modelling a weapon determines that it has impacted upon, or detonated near, an entity modelled by this federate, it will advise the external entity federate via a weapon impact/detonation notification message.

The federate will receive execution management messages and respond in accordance with the concept for execution management. Execution management error messages may be issued.

5.4.8.2 Other data required

The external entity federate may be considered as an aggregation of all the preceding federates and may require similar additional data. These data include environmental data required to compute the performance of sensors, environmental data required to compute the motion of the entity and data describing the physical construction of the entity and arrangement of systems within it.

5.4.9 Execution manager federate

A typical execution management federate is illustrated in Figure 5.4.9a.

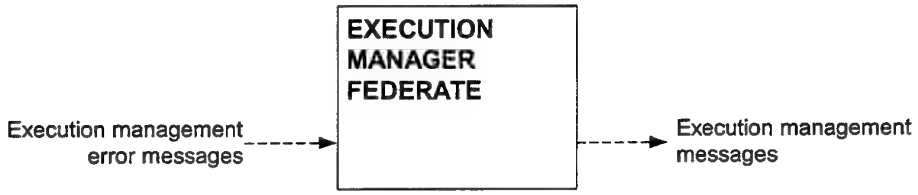


Figure 5.4.9a: The data input to, and output from, a typical execution manager federate.

5.4.9.1 Typical behaviour

The execution manager federate controls the flow of federation execution in accordance with the concept for execution management (see section 8). It achieves this through the issue of execution management messages. In addition, it receives execution management error messages in the event that a federate is unable to respond to execution management messages.

5.4.9.2 Other data required

The execution manager federate requires an execution management script file as input data. This file details the federates participating in a federation execution, the set of entities that define the initial conditions for the federation execution and any entities that are injected into the federation at specified times (see *Execution Management in the Virtual Ship Architecture* [2]).

5.4.10 Utility federate

A typical utility federate is illustrated in Figure 5.4.10a.

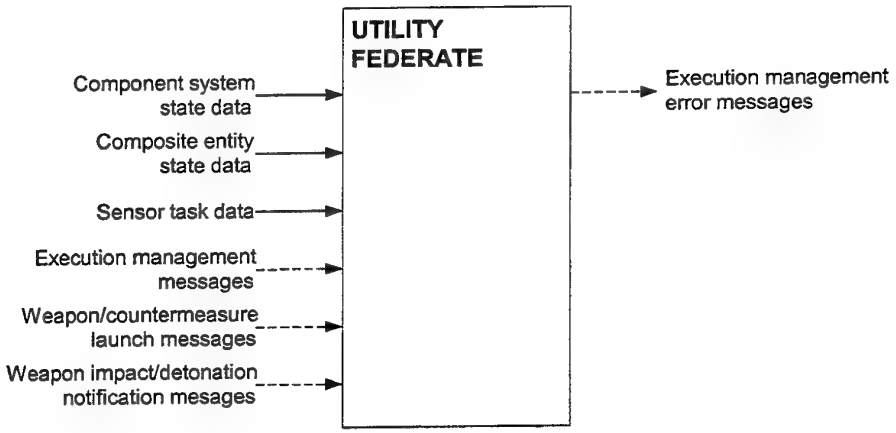


Figure 5.4.10a: The data input to, and output from, a typical utility federate.

5.4.10.1 Typical behaviour

A utility federate provides functionality such as stealth viewing, data logging and data analysis. To perform these functions requires receipt of composite entity state data, component system state data and sensor task data. The latter will enable, for example, sensor coverages to be viewed on a stealth display.

Weapon/countermeasure launch messages and weapon impact/detonation notification messages are also received so that these may be recorded or rendered on a display.

Although not precluded, system control messages, track data and system damage notification messages are typically not required by this federate (see section 5.5).

The federate will receive execution management messages and respond in accordance with the concept for execution management. Execution management error messages may be issued.

5.4.10.2 Other data required

Nil

5.5 Data flows within the federation

Figure 5.5a shows the typical data flows within the federation. Those federates that constitute components of a Virtual Ship have been grouped and the notation indicates the possibility of multiple instances of both certain types of federates and the Virtual Ship.

The figure illustrates a key point. There are certain data items that remain within the confines of the Virtual Ship. These are system control messages, system damage notification messages and track data. These are not exchanged with federates

external to the Virtual Ship[2]. Hence the diagram illustrates the concept of aggregation within the Virtual Ship. A single instance of the Virtual Ship, although composed of many federates, interacts with the external entities and other Virtual Ships, in the sense of data exchange, as if it were a single federate. Hence, any entity within the Virtual Ship federation may be a single federate or may be decomposed into multiple federates that represent its constituent parts. The notion of an external entity federate provides a logical grouping of all federates that represent constituent parts of a ship.

[2] It is, however, possible to conceive of a utility federate that recorded all FOM data exchanged by the federates.

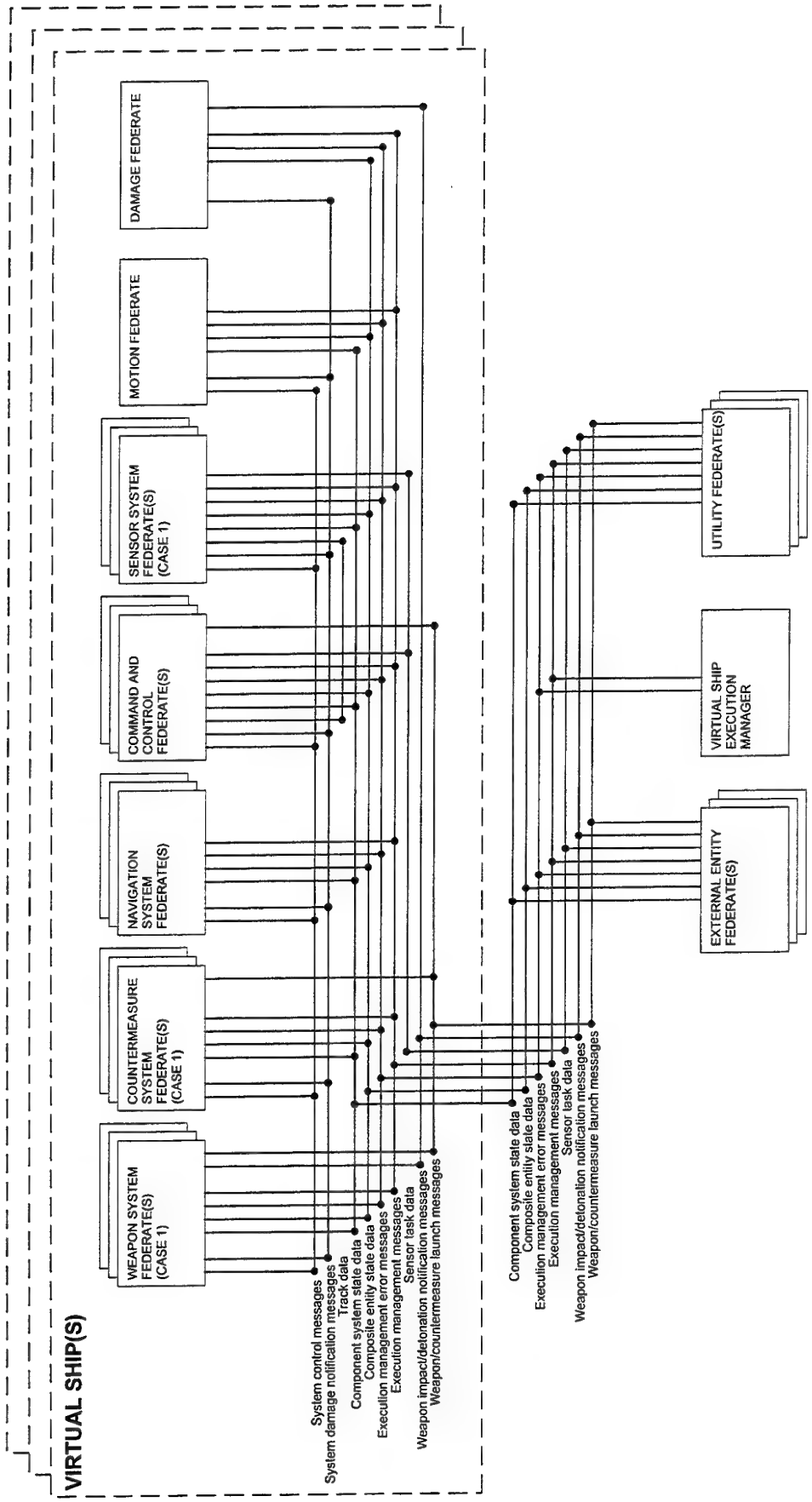


Figure 5.5a: Data exchange within a Virtual Ship federation.

6. The Federation Object Model view of the Virtual Ship Architecture

6.1 Introduction

The Virtual Ship Federation Object Model (VS-FOM) provides an object view of the data exchanged between federates. Data items are grouped together as attributes of objects, in the case of data with persistence, and as parameters of interactions, in the case of data lacking persistence. These objects typically correspond to entities within the physical domain and the interactions correspond to interactions between these entities.

The object and interaction classes are arranged in hierarchical structures, with an inheritance relationship in effect. Subclasses represent specialisations of a given class. They inherit the attributes of the superclass and may have additional attributes that define their specialised properties. However, it must be borne in mind that this specialisation is sometimes of a conceptual nature. The subscription and publication paradigm within the HLA provides a mechanism for data filtering. Under some circumstances it is appropriate that subclasses have no attributes additional to those of their superclass. The existence of the subclass permits selective subscription to objects that define an abstract specialisation of the superclass.

The key components of the FOM are the Object Class Structure Table and the Interaction Class Structure Table. These are described in this section. A graphical approach is used to illustrate the inheritance relationships within the VS-FOM. In section 7, the relationship between the federate view and the Federation Object Model view is described through reference to an example federation.

6.2 The Object Class Structure Table

The Object Class Structure Table is shown as Table 6.1. There are four branches: CompositeEntity, ComponentEntity, SensorTask and Track.

Recall the notation of P, S or N after each class name. The notation P indicates that the object class may be published within the federation and S indicates that the object class may be subscribed. The notation N indicates that the object class may neither be published nor subscribed.

Table 6.2a: The Object Class Structure Table

CompositeEntity (N)	Air (PS)	
	SeaSurface (PS)	
	SubSurface (PS)	
ComponentEntity (N)	CommandAndControlSystem (PS)	
	CountermeasureSystem (PS)	
	NavigationSystem (PS)	
	SensorSystem (N)	ESMSystem (PS)
		IRSystem (PS)
		IFFSystem (PS)
		RadarSystem (PS)
		SonarSystem (PS)
	WeaponSystem (PS)	
SensorTask (N)	ESMTask (PS)	
	IRTask (PS)	
	IFFTask (PS)	
	RadarTask (PS)	
	SonarTask (PS)	
Track (N)	AbsoluteTrack (N)	AbsoluteSystemTrack (PS)
		AbsoluteFusedTrack (PS)
	RelativeTrack (N)	RelativeSystemTrack (PS)
		RelativeFusedTrack (PS)
		RelativeESMTrack (PS)
		RelativeIRTrack (PS)
		RelativeIFFTrack (PS)
		RelativeRadarTrack (PS)
		RelativeSonarTrack (PS)

6.2.1 The CompositeEntity branch

A class diagram illustrating the CompositeEntity branch of the Object Class Structure is shown as Figure 6.2.1a.

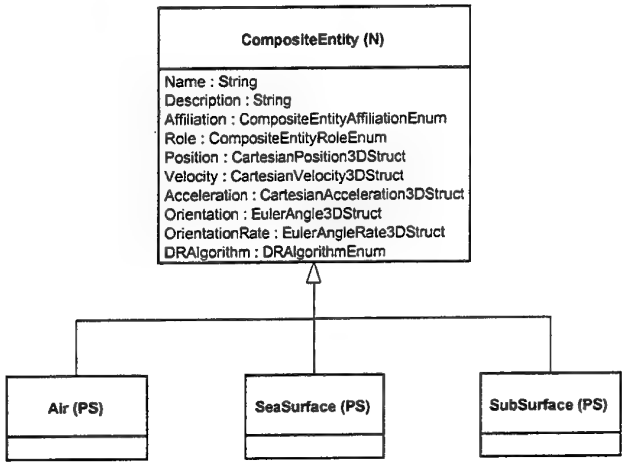


Figure 6.2.1a: The CompositeEntity branch of the Object Class Structure.

The basic entity considered to exist within the problem space is a composite entity. A composite entity represents a complete entity, such as a ship, submarine, aircraft , missile or countermeasure. Instances of the CompositeEntity object class facilitate exchange between federates of data describing composite entities.

The following attributes of the CompositeEntity class permit identification of instances:

Attribute	Description
Name	This attribute is used to specifically identify a particular instance of a CompositeEntity. The name attribute is composed of a concatenation of terms, each of which conveys a well-defined characteristic of the entity (see section 11).
Description	This attribute provides a user defined description of a CompositeEntity. The typical use of this attribute is to provide a description of the CompositeEntity that may be displayed in tools that illustrate the state of a scenario, such as a stealth viewer.
Affiliation	Names the organisation the CompositeEntity is a member of. This attribute may take values from the enumerated datatype CompositeEntityAffiliationEnum. A typical use of this attribute is to provide information concerning a CompositeEntity that may be displayed in tools that illustrate the state of a scenario, such as a stealth viewer.
Role	Describes the role currently being executed by a CompositeEntity. This attribute may take values from the enumerated datatype CompositeEntityRoleEnum. The typical use of this attribute is to provide information about the role of the CompositeEntity that may be displayed in tools that illustrate the state of a scenario, such as a stealth viewer.

The following attributes specify the kinematics (see [1]) of the composite entity:

Attribute	Description
Position	The position of a reference point of the CompositeEntity. The position is given with respect to the WGS84 Cartesian coordinate system.
Velocity	The velocity of the CompositeEntity. The velocity components are given with respect to the WGS84 Cartesian coordinate system.
Acceleration	The acceleration of the CompositeEntity. The components of the acceleration are given with respect to the WGS84 Cartesian coordinate system.
Orientation	The orientation of a coordinate system fixed to the CompositeEntity with respect to the WGS84 Cartesian coordinate system. The origin of this coordinate system is given by the Position attribute.
OrientationRate	The rate of change of the Euler angles that define the orientation of a coordinate system fixed to the CompositeEntity with respect to the WGS84 Cartesian coordinate system.
DRAlgorithm	Specifies the algorithm used to dead reckon (project forward in time) the kinematic attributes of the CompositeEntity.

There are three subclasses of the CompositeEntity class, detailed in the following table. These have no additional attributes. However, they allow federates to selectively subscribe to instances of the CompositeEntity class in various physical domains. This will provide benefits when certain federates only model within a single domain.

Class Name	Description
Air	Instances of this object class describe those entities that are commonly recognised as existing predominantly in the air domain. This includes aircraft, missiles and other entities launched into the air, such as the Nulka active decoy.
SeaSurface	Instances of this object class describe those entities that are commonly recognised as existing in the sea surface domain. This principally includes all watercraft that do not have the capability to submerge.
SubSurface	Instances of this object class describe those entities that are commonly recognised as existing in the sub-surface domain. This principally includes submarines, irrespective of whether they are submerged or not, and torpedoes.

6.2.2 The ComponentEntity branch

A class diagram illustrating the ComponentEntity branch of the Object Class Structure is shown as Figure 6.2.2a.

Component entities are considered part of a composite entity and data describing them are provided as attributes of ComponentEntity object instances. Component entities are typically considered to be the systems that make up a complex, or composite, entity. The attributes of the ComponentEntity class are as follows:

Attribute	Description
ComponentName	This attribute provides a description of a ComponentEntity. This attribute is composed of a concatenation of terms, each of which conveys a well-defined characteristic of the entity. (See section 12.)
RelativePosition	The position of the ComponentEntity relative to the reference point of the parent CompositeEntity. The position is given with respect to the coordinate system fixed to the parent CompositeEntity.
RelativeOrientation	The orientation of a coordinate system fixed to the ComponentEntity relative to the coordinate system fixed to the parent CompositeEntity.
ParentCompositeEntityObjectInstanceName	The object instance name of the CompositeEntity that is the parent of the ComponentEntity. This attribute is used to identify which CompositeEntity kinematic attributes should be used in conjunction with the RelativePosition and RelativeOrientation attributes to derive the kinematic attributes of the ComponentEntity.

The ParentCompositeEntityObjectInstanceName attribute is critical. When an object instance is registered with the RTI, either the user specifies a string that uniquely identifies the object instance throughout the federation, or the RTI assigns such a string. This attribute is used by the federates in the federation to associate the component entity with its parent composite entity.

The subclasses of the ComponentEntity are detailed as follows:

Class Name	Description
CommandAndControlSystem	Instances of this object class represent component systems that have command and control functionality. Examples might include data fusion systems, threat evaluation and weapon assignment systems, fire control, etc.
CountermeasureSystem	Instances of this object class represent component systems that have countermeasure functionality, such as a jammer, or that deploy entities that perform a countermeasure function, such as a chaff launcher.
NavigationSystem	Instances of this object class represent component systems that have navigation functionality.
SensorSystem	Instances of this object class represent component systems that have sensing functionality.
WeaponSystem	Instances of this object class represent component systems that deploy weapons, whether they be guided or otherwise.

These do not possess any additional attributes and are introduced to facilitate selective subscription to, and discovery of, particular types of component entity.

The SensorSystem subclass has been further decomposed through introduction of a number of subclasses. Again, these have been introduced to facilitate selective subscription and discovery.

Class Name	Description
ESMSystem	Instances of this object class represent sensor systems that passively detect entities that emit electromagnetic radiation, typically from radars and radio antennae.
IRSystem	Instances of this object class represent sensor systems that passively detect emissions from entities in the infrared part of the electromagnetic spectrum.
IFFSystem	Instances of this object class represent sensor systems that utilise the IFF (interrogate friend or foe) system to interrogate entities, typically in the air domain.
RadarSystem	Instances of this object class represent sensor systems that utilise radar to detect entities.
SonarSystem	Instances of this object class represent sensor systems that actively and/or passively detect entities through their sound scattering and/or emitting properties.

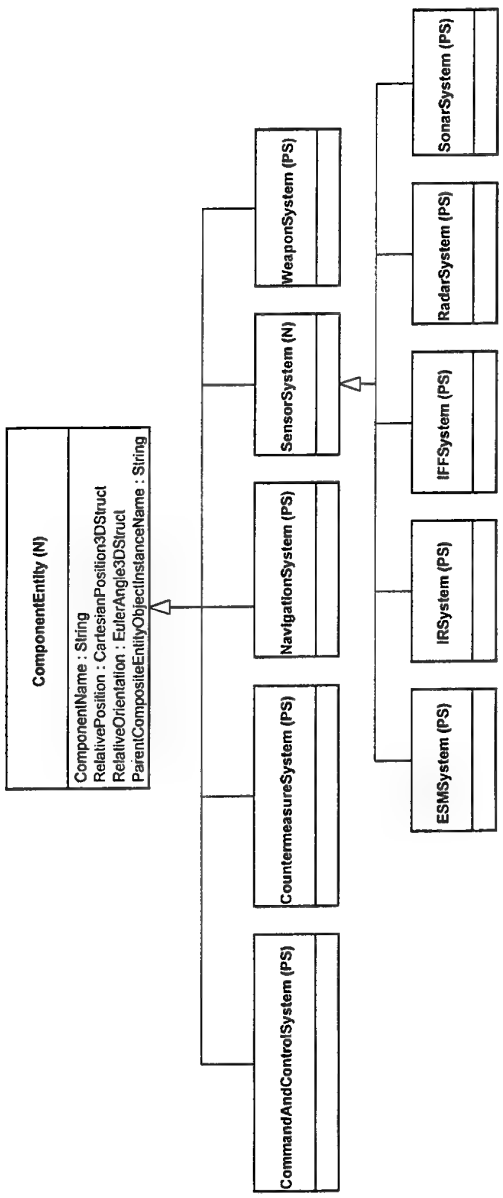


Figure 6.2.2a: The ComponentEntity branch of the Object Class Structure.

6.2.3 The SensorTask branch

A class diagram illustrating the SensorTask branch of the Object Class Structure is shown as Figure 6.2.3a.

A modern sensor may simultaneously perform a number of tasks. Instances of the SensorTask object convey data describing these tasks to the federation. Sensor tasks are associated with the component entities that represent sensors.

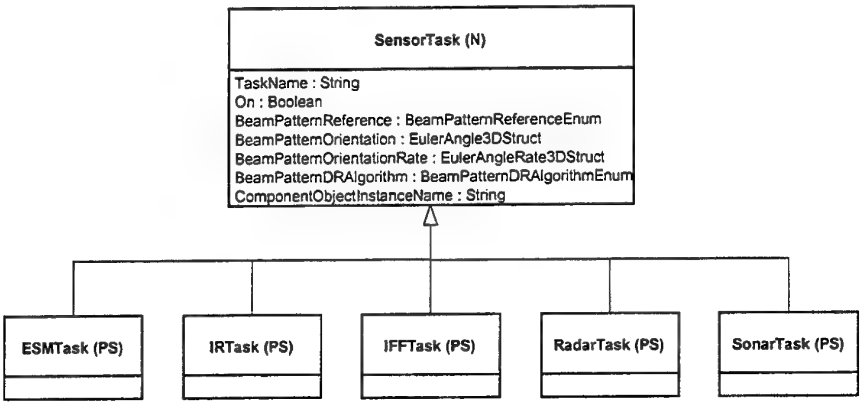


Figure 6.2.3a: The SensorTask branch of the Object Class Structure.

The attributes of the SensorTask object class are as follows:

Attribute	Description
TaskName	Specifies the task performed by a sensor. A single sensor may perform any number of tasks.
On	This is a boolean attribute which is true if the task is currently being executed and false otherwise.
BeamPatternReference	Specifies the coordinate system with respect to which the beam pattern orientation is given. The reference coordinate system may be the WGS84 Cartesian coordinate system, a local geodetic coordinate system or the coordinate system fixed to the sensor's parent composite entity.
BeamPatternOrientation	The orientation of a coordinate system fixed to the beam pattern, given with respect to the coordinate system specified by the BeamPatternReference attribute.
BeamPatternOrientationRate	The rate of change of the Euler angles that define the orientation of a coordinate system fixed to the beam pattern, given with respect to the coordinate system defined by the BeamPatternReference attribute.
BeamPatternDRAlgorithm	The algorithm used to dead reckon the Euler angles that specify the orientation of the beam pattern associated with this task.
ComponentObjectInstanceName	The object instance name of the ComponentEntity.SensorSystem that represents the

	sensor system that this task is associated with.

The ComponentObjectInstanceName attribute is used to identify the particular component entity that a sensor task is associated with.

There are five subclasses of the SensorTask class. These are

Class Name	Description
ESMTask	Instances of this object class describe those tasks performed by an electronic support sensor.
IRTask	Instances of this object class describe those tasks performed by an infrared sensor.
IFFTask	Instances of this object class describe those tasks performed by an IFF system.
RadarTask	Instances of this object class describe those tasks performed by a radar sensor.
SonarTask	Instances of this object class describe those tasks performed by a sonar sensor.

These classes have no additional attributes but facilitate selective subscription and discovery of sensor tasks.

6.2.4 The Track branch

A class diagram illustrating the Track branch of the Object Class Structure is shown as Figure 6.2.4a.

Instances of the Track object convey data concerning the estimated position of entities as determined by sensors. The Track class attributes consist of those data items common to all representations of tracks. These are detailed as follows:

Attribute	Description
Time	The time for which the current track attributes are valid.
TimeOfInitiation	The time at which the track was initiated.
Valid	A boolean flag with value true if the track is valid. The flag indicates whether the data describing the track can be relied upon for display or further processing.
Lost	A boolean flag with value true if the track has been lost. A track that has been lost may still have value as it provides an indicator of an entity's presence and motion at some time in the past.
Description	A user defined description of the track.
TrackNumber	The track number assigned by the sensor generating the track. A track is uniquely defined by a combination of this track number and the RTI name of the entity that generates it, usually a sensor system.
TargetObjectInstanceNameSet	The object instance names of the

Attribute	Description
	CompositeEntity objects that may give rise to this track. Ideally, there is only one such object associated with the track. However, given that some sensors may not be able to resolve nearly co-located entities, there may be more than one object instance associated with it.
TrackGeneratorObjectInstanceName	The object instance name of the entity that generates the track. This will typically be a ComponentEntity.SensorSystem object instance.
TrackQuality	Provides a somewhat subjective measure of the reliability of the track data. This attribute may take values from the enumerated datatype TrackQualityEnum.
PlatformIDClassification	An estimate of the identity of the tracked platform. This attribute consists of a list of possible platforms along with a measure of the confidence in each case.
ThreatClassification	A classification of the intention of the platform that is being tracked, for example hostile or friendly. This attribute may take values from the enumerated datatype TrackedPlatformThreatClassificationEnum.

There are two subclasses of the Track class. The attributes of these subclasses give the position of the tracked entity in either absolute coordinates or relative coordinates.

Class Name	Description
AbsoluteTrack	This class represents tracks defined in terms of the WGS84 Cartesian coordinate system.
RelativeTrack	This class represents tracks defined relative to the position of the generating sensor.

The attributes of the AbsoluteTrack object class are detailed in the following:

Attribute	Description
PositionValid	A flag with value true if the Position attribute can be relied upon.
Position	An estimate of the position of the tracked object given as a position vector with respect to the WGS84 Cartesian coordinate system.
PositionError	The error with which the position is given, using the same coordinate system.
VelocityValid	A flag with value true if the Velocity attribute can be relied upon.
Velocity	An estimate of the velocity of the track given as a vector with respect to the WGS84 Cartesian coordinate system.
VelocityError	The error with which the velocity is given, using the same coordinate system.

The attributes of the RelativeTrack object class are as follows:

Attribute	Description
FootpointPosition	The position of the point with respect to which the position of the track is given. This is typically the position of the sensor generating the relative track.
FootpointPositionError	The error with which the footpoint position is given, using the same coordinate system.
BearingValid	A flag with value true if the Bearing attribute can be relied upon.
Bearing	An estimate of the bearing of the tracked object as measured from the footpoint relative to Geodetic North.
BearingError	The error with which the bearing is given.
ElevationValid	A flag with value true if the Elevation attribute can be relied upon.
Elevation	An estimate of the elevation of the tracked object as measured from the footpoint relative to the plane perpendicular to the local gravitational field.
ElevationError	The error with which the elevation is given.
RangeValid	A flag with value true if the Range attribute can be relied upon.
Range	An estimate of the range of the tracked object as measured in a straight line from the footpoint.
RangeError	The error with which the range is given.
RangeRateValid	A flag with value true if the RangeRate attribute can be relied upon.
RangeRate	An estimate of the rate of change of the range of the tracked object as measured along a straight line from the footpoint.
RangeRateError	The error with which the range rate is given.

The AbsoluteTrack and RelativeTrack classes are further specialised in the form of subclasses. The subclasses of the AbsoluteTrack class are as follows:

Class Name	Description
AbsoluteSystemTrack	This class represents system tracks that are defined in terms of the WGS84 Cartesian coordinate system. A system track is generally formed using data from one or more sensor tracks.
AbsoluteFusedTrack	Instances of this object class represent tracks generated by fusing two of more sources of track data and use the WGS84 Cartesian coordinate system.

The subclasses of the RelativeTrack are as follows:

Class Name	Description
RelativeSystemTrack	This class represents system tracks that are defined in terms of the position of the composite entity that generates it. A system track is generally formed using data from one or more sensor tracks.
RelativeFusedTrack	Instances of this object class represent tracks generated by fusing two of more sources of track data and given relative to a footpoint.
RelativeESMTrack	Instances of this object class represent tracks generated by

Class Name	Description
	electronic support sensors.
RelativeIRTrack	Instances of this object class represent tracks generated by infrared sensors.
RelativeIFFTrack	Instances of this object class represent tracks generated by IFF sensors.
RelativeRadarTrack	Instances of this object class represent tracks generated by radar sensors.
RelativeSonarTrack	Instances of this object class represent tracks generated by sonar sensors.

The attributes of the AbsoluteSystemTrack are as follows:

Attribute	Description
SourceTracks	A list of the tracks that have been combined to create the system track. Many different techniques might be used for this process.

The attributes of the AbsoluteFusedTrack are as follows:

Attribute	Description
SourceTracks	A list of the tracks that have been combined to create the fused track. Many different techniques might be used for this process.

The attributes of the RelativeSystemTrack are as follows:

Attribute	Description
SourceTracks	A list of the tracks that have been combined to create the system track. Many different techniques might be used for this process.

The attributes of the RelativeFusedTrack are as follows:

Attribute	Description
SourceTracks	A list of the tracks that have been combined to create the fused track. Many different techniques might be used for this process.

The RelativeESMTrack has no additional attributes. The RelativeIRTrack class has the following attributes:

Attribute	Description
IntensityValid	A flag with value true if the intensity attribute can be relied upon.
Intensity	An estimate of the IR intensity of the tracked entity.
IntensityError	The error with which the intensity is given.

The RelativeIFFTrack class has the following attributes:

Attribute	Description
Code	The IFF code received from a tracked entity.
AltitudeValid	A flag with value true if the Altitude attribute can be relied upon.
Altitude	An estimate of the altitude of the tracked entity above sea level.

AltitudeError	The error with which the altitude is given.

The RelativeRadarTrack has the following attributes:

Attribute	Description
RCSValid	A flag with value true if the RCS (radar cross-section) attribute can be relied upon.
RCS	An estimate of the RCS of the tracked entity.
RCSError	The error with which the RCS is given.
SNRValid	A flag with value true if the SNR (single-to-noise ratio) attribute can be relied upon.
SNR	An estimate of the SNR of the signal received from the tracked entity.
SNRError	The error with which the SNR is given.

The RelativeSonarTrack has the following attributes:

Attribute	Description
SNRValid	A flag with value true if the SNR attribute can be relied upon.
SNR	An estimate of the SNR of the signal received from the tracked entity.
SNRError	The error with which the SNR is given.
FrequencyValid	A flag with value true if the Frequency attribute can be relied upon.
Frequency	An estimate of the frequency of the signal received from the tracked entity.
FrequencyError	The error with which the frequency is given.
DopplerValid	A flag with value true if the Doppler attribute can be relied upon.
Doppler	An estimate of the doppler shift associated with the tracked entity.
DopplerError	The error with which the doppler is given.
CourseValid	A flag with value true if the Course attribute can be relied upon.
Course	An estimate of the course of the tracked entity.
CourseError	The error with which the course is given.
SpeedValid	A flag with value true if the Speed attribute can be relied upon.
Speed	An estimate of the speed of the tracked entity.
SpeedError	The error with which the speed is given.
BearingAmbiguity	A flag with value true if the bearing of the tracked entity is ambiguous.
AmbiguousBearing	An estimate of the ambiguous bearing of the tracked entity.
AmbiguousBearingError	The error with which the ambiguous bearing is given.

Figure 6.2.4a: The Track branch of the Object Class Structure.

6.3 The Interaction Class Structure Table

The Interaction Class Structure Table is shown as Table 6.3a. There are two branches: ExecutionManagement and SystemControl.

Recall the notation of I, R, S or N after each class name. The notation I indicates that the interaction class may be published (initiated) within the federation and R indicates that the object class may be subscribed (react). The notation S also indicates that an interaction may be subscribed but that it will be sensed, not reacted to. The notation N indicates that the interaction class may neither be published nor subscribed.

Table 6.3a: The Object Interaction Table.

ExecutionManagement (N)	CreateCompositeEntity (IR)	
	CreateComponentEntity (IR)	
	SetScenarioDescription (IR)	
	SetRandomNumberSeed (IR)	
	ExecutionManagementError (IR)	
SystemControl (N)	PropulsionSystemControl (N)	SetPropulsionSystemAttribute (IR)
		AckSetPropulsionSystemAttribute (IR)

6.3.1 The ExecutionManagement branch

The ExecutionManagement class is an abstract class introduced to group interactions that support execution management (see section 8). It is illustrated in Figure 6.3.1a and has the following subclasses:

Interaction	Description
CreateCompositeEntity	Provides an instruction to a federate to create an instance of a CompositeEntity object.
CreateComponentEntity	Provides an instruction to a federate to create an instance of a ComponentEntity object.
SetScenarioDescription	Provides to a set of managed federates a description that uniquely identifies a particular Virtual Ship scenario.
SetRandomNumberSeed	Provides to a set of managed federates a random number seed to be used, if required, to pseudo-randomly adjust certain scenario characteristics.
ExecutionManagementError	Provides to the Execution Manager an indication of why a federate has been unable to comply with an execution management instruction.

Figure 6.3.1a: The ExecutionManagament branch of the Interaction Class Structure.

The CreateCompositeEntity interaction class has the following parameters:

Parameter	Description
ClassName	Specifies the subclass of the CompositeEntity object that is to be created. It will take on values of Air, SeaSurface or SubSurface.
ObjectInstanceName	Specifies the object instance name of the CompositeEntity object that is to be created.
Federate	Specifies the name of the federate that is to create the CompositeEntity object.
ConfigurationFile	Specifies the name of the configuration file to be used by the federate that creates the CompositeEntity object. It is assumed that this file is local to the federate.
AttributeNames	Provides a list of the attributes of the CompositeEntity object that are to be initialised with values other than default and specified by the AttributeValues parameter.
AttributeValues	Provides a list of the values of the attributes of the CompositeEntity object that are specified by the AttributeNames parameter. These are non-default attribute values.

The CreateComponentEntity interaction class has the following parameters:

Parameter	Description
ClassName	Specifies the subclass of the ComponentEntity object that is to be created.
ObjectInstanceName	Specifies the object instance name of the ComponentEntity object that is to be created.
Federate	Specifies the name of the federate that is to create the ComponentEntity object.
ConfigurationFile	Specifies the name of the configuration file to be used by the federate that creates the ComponentEntity object. It is assumed that this file is local to the federate.
ParentObjectInstanceName	Provides the object instance name of the CompositeEntity object that is the parent of the ComponentEntity object to be created.
AttributeNames	Provides a list of the attributes of the ComponentEntity object that are to be initialised with values other than default and specified by the AttributeValues parameter.
AttributeValues	Provides a list of the values of the attributes of the ComponentEntity object that are specified by the AttributeNames parameter. These are non-default attribute values.

The SetScenarioDescription interaction has the following parameter:

Parameter	Description
Description	Provides a textual description of a particular scenario. It is typically used in the event that a scenario is repeated during a single federation execution, in order to uniquely identify each one.

The SetRandomNumberSeed interaction has the following parameter:

Parameter	Description
Seed	Provides to a set of managed federates a random number seed to be used, if required, to pseudo-randomly adjust certain scenario characteristics.

The ExecutionManagementError interaction has the following parameters:

Parameter	Description
Error	Indicates the type of execution management error from an enumerated list.
Information	Provides additional information concerning the failure of a federate to carry out an execution management instruction.

6.3.2 The SystemControl branch

The SystemControl class is an abstract class introduced to group interactions that support the exercise of control over component systems. It is illustrated in Figure 6.3.2a.

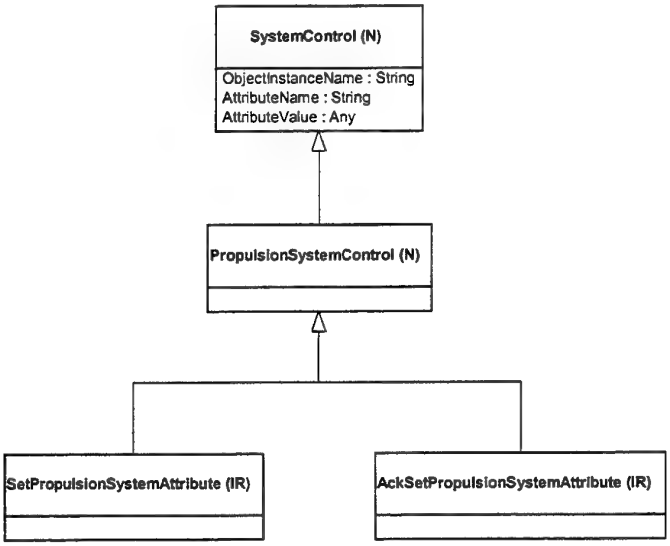


Figure 6.3.2a: The SystemControl branch of the Interaction Class Structure.

The underlying methodology for the exercise of control is that a request to change an attribute value is sent, in the form of an interaction, to a federate modelling a system and this federate acknowledges receipt of the message and provides an indication of

its compliance or otherwise. The SystemControl interaction class has the following parameters:

Parameter	Description
ObjectInstanceName	Identifies the ComponentEntity object instance that is the target for the interaction.
AttributeName	Specifies the attribute that is to be changed in response to this interaction. This need not correspond to an attribute of the ComponentEntity class.
AttributeValue	Specifies the new value for the attribute.

The SystemControl interaction class has the following subclass.

Interaction	Description
PropulsionSystemControl	Provides a grouping of interactions that support the exercise of control over propulsion systems.

The PropulsionSystemControl interaction class has no parameters. It facilitates selective publication and subscription of interactions associated with the control of propulsion systems.

The PropulsionSystemControl interaction class has the following subclasses:

Interaction	Description
SetPropulsionSystemAttribute	Provides an instruction to a federate representing a propulsion system to set the specified attribute to a given value.
AckSetPropulsionSystemAttribute	Provides an acknowledgment of receipt of the SetPropulsionSystemAttribute interaction. In the event that the SetPropulsionSystemAttribute interaction cannot be complied with, the AttributeValue parameter conveys the current value of the attribute identified by AttributeName.

7. The relationship between the federate and FOM views

7.1 Introduction

Understanding the Virtual Ship Architecture is enhanced through consideration of the relationship between the federate view and FOM view. To explore this relationship we consider an example federation. Each of the federates within this federation is analysed to identify their specific data requirements and how these are satisfied through publication and subscription of various FOM object classes.

7.2 The scenario and its implementation as a federation

The federation used to illustrate the relationship between the federate and FOM views of the Virtual Ship Architecture has been constructed to support research in sensor data fusion. What is required is a simulation of track data as output by multiple sensors on board a warship. To achieve this objective, the scenario consists of a single ship coming under attack by a single anti-ship missile, launched from over the horizon. The ship is equipped with radar, ESM (electronic support measures) and IRST (infrared search and track) sensors, which generate track data. A fusion system takes these track data and fuses them to form a tactical picture.

The federation developed to implement this scenario consists of nine federates. These are:

1. Virtual Ship Execution Manager (VSEM) - An implementation of the execution management concept (see section 8).
2. Motion - Models the motion of the warship.
3. Helm (Navigation federate) - Provides for interactive manoeuvring of the warship.
4. Radar (Sensor federate) - Models the performance of the ship's radar.
5. ESM (Sensor federate) - Models the performance of the ship's ESM sensor.
6. IRST (Sensor federate) - Models the performance of the ship's IRST sensor.
7. Fusion (Command and control federate) - Fuses track data from the ship's sensors to generate a tactical picture.
8. Missile (External entity federate) - Models the anti-ship missile.
9. Virtual Ship Simulation Display (VSSD) (Utility federate) - Provides a plan view display of the scenario.

Note that the federates have been identified in terms of the taxonomy of federates introduced in section 5.2. This federation is referred to as the Sensor Fusion Federation and is illustrated in Figure 7.2a. To generate this figure, Figure 5.2a has been modified by shading out missing federates. It illustrates that this federation consists of only a subset of those conceived as part of the Virtual Ship. Although the scenario consists of a single ship and missile, each entity may be replicated to create a more complex scenario.

The remainder of this section will consider each federate in turn, with particular emphasis on their publications and subscriptions as the mechanism by which they

provide, and receive, the data required for them to interoperate. Two types of diagram are used to aid this description.

One is a traditional collaboration diagram that illustrates the relationship between FOM objects that a federate either publishes or subscribes and is referred to as a FOM object collaboration diagram. Lines connecting the objects indicate a collaboration relationship and text at each end of the line describes the role that the object plays, from the viewpoint of the other. In addition, the diagrams illustrate cardinality relationships, that is the number of possible instances of an object that are associated with the collaborating object. It is particularly noted that the notation 0..n indicates zero or more, and that repetition of 0..n throughout a single diagram does not imply a common number n across different relationships.

The second diagram extends the collaboration diagram to illustrate the collaboration between federates and is referred to as a federate collaboration diagram. Lines joining federates indicate the collaboration through the exchange of data. This diagram supplements the first by illustrating the source of subscribed data and the destination of published data.

Note that the FOM object collaboration diagram reflects object class publications and subscriptions, whereas the federate collaboration diagram additionally reflects interactions.

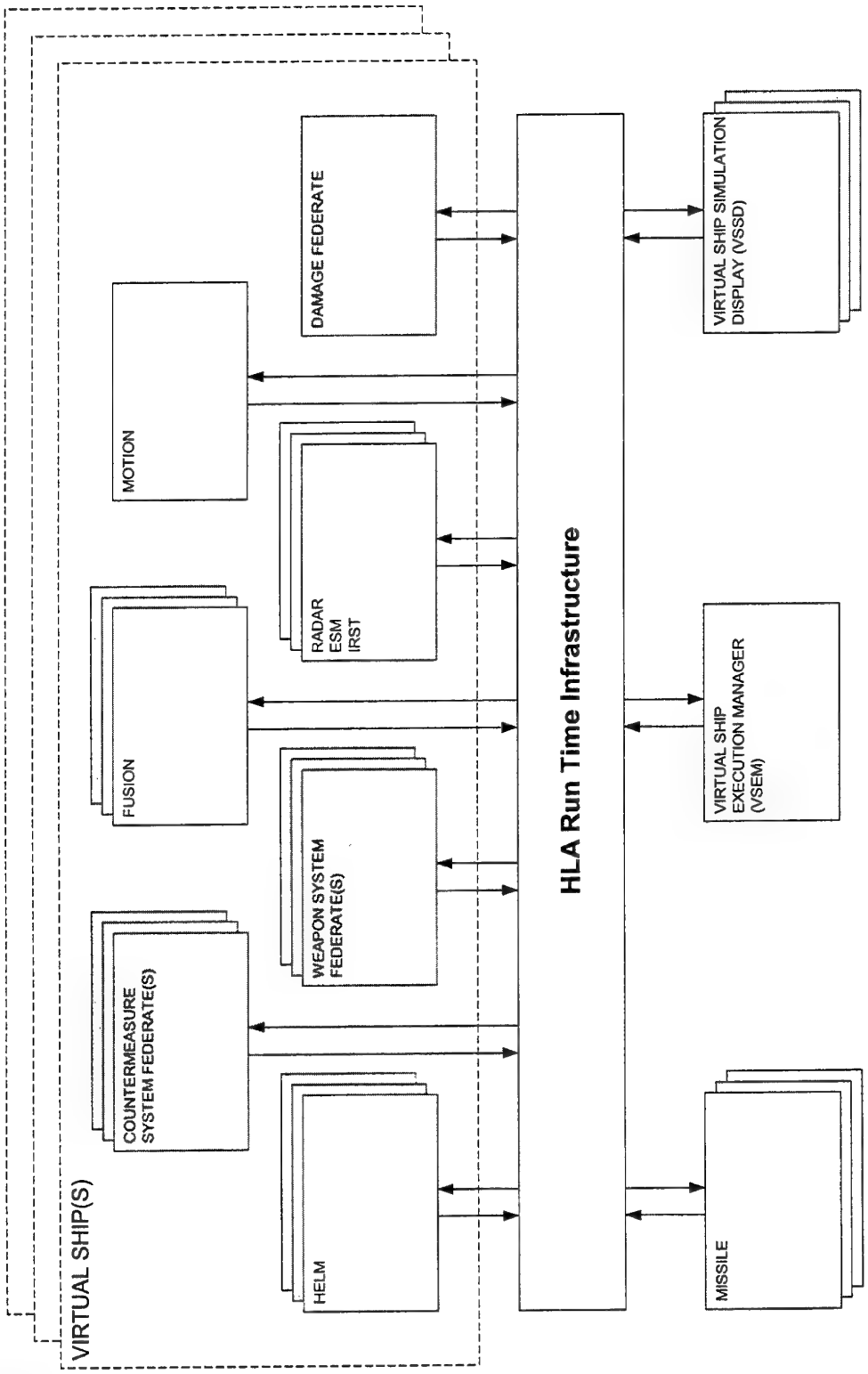


Figure 7.2a: The Sensor Fusion federation.

7.3 The Motion federate

The Motion federate models the motion of multiple warships. It is additionally capable of representing entities in the air and subsurface domains. The FOM object collaboration diagram is shown as Figure 7.3a. It illustrates that this federate publishes instances of the CompositeEntity.SeaSurface object class, which represent individual warships. The federate may also publish CompositeEntity.Air and CompositeEntity.SubSurface object instances in the event that it is used to represent entities in these domains.

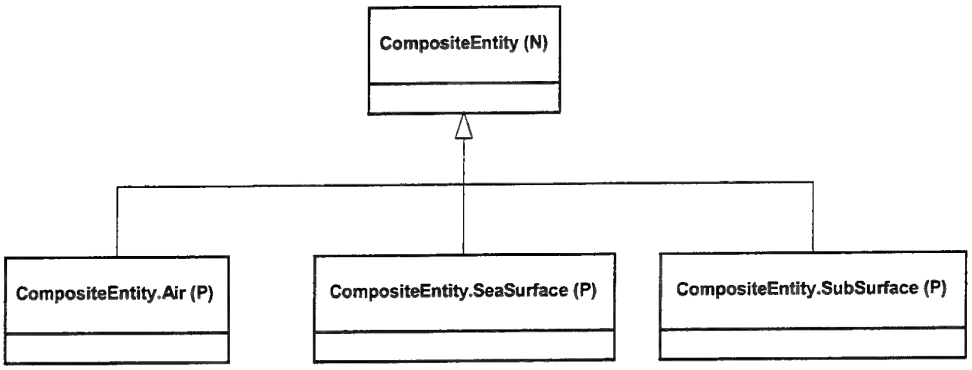


Figure 7.3a: FOM object collaboration diagram for the Motion federate.

The federate collaboration diagram for the Motion federate is illustrated in Figure 7.3b. This diagram illustrates the flow of data between the Motion federate and the others in the federation. The CompositeEntity.SeaSurface object instances published by the Motion federate are subscribed to by the Radar, ESM, IRST, Fusion, Helm, Missile and VSSD federates. It is through these subscriptions that these federates are informed of the existence, state and kinematic attributes of the warship and can therefore compute their interaction with it.

The Motion federate collaborates with the VSEM federate through the mechanism of interactions, not object attribute updates. Interactions that are subclasses of the ExecutionManagement class[3] are sent from the VSEM to the Motion federate in order to manage the federation execution. The Motion federate sends ExecutionManagement.ExecutionManagementError interactions in the event that an instruction from the VSEM cannot be complied with.

The SystemControl.PropulsionSystemControl.SetPropulsionSystemAttribute interaction is sent to the Motion federate from the Helm federate to set the course and speed of an entity modelled by the federate. Receipt of this interaction is acknowledged through the Motion federate sending the SystemControl.PropulsionSystemControl.AckSetPropulsionSystemAttribute interaction to the Helm federate.

[3] Excluding the ExecutionManagementError interaction that is sent from the Motion federate to the VSEM.

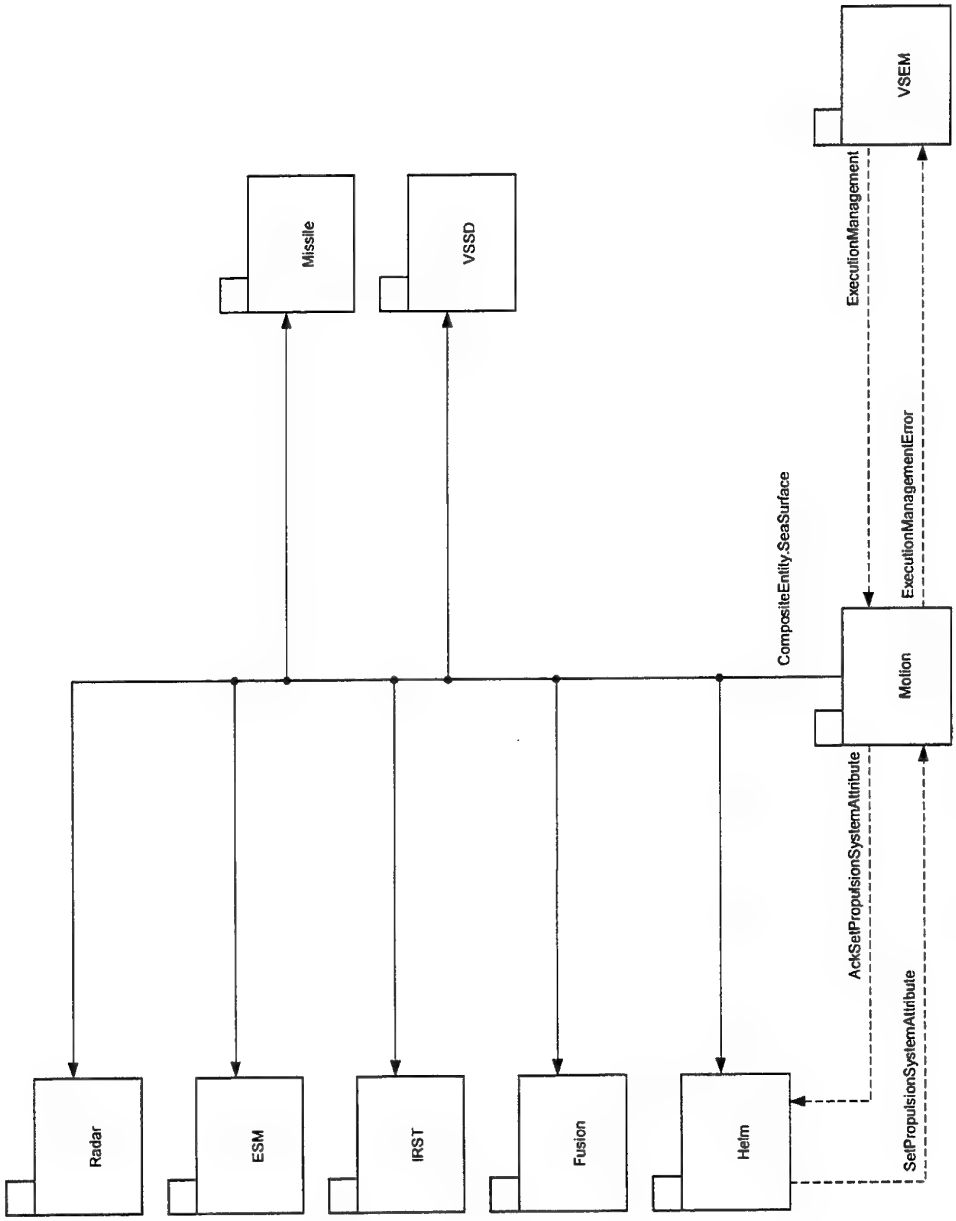


Figure 7.3b: Federate collaboration diagram for the Motion federate.

7.4 The Helm federate

The Helm federate provides a graphical user interface through which the warship can be manoeuvred. It provides for selection of the speed and course of the vessel. The FOM object collaboration diagram is shown as Figure 7.4a. It illustrates that this federate publishes an instance of the ComponentEntity.NavigationSystem object class, which represents the navigation system. The federate subscribes to the CompositeEntity.SeaSurface object class, an instance of which represents the warship being manoeuvred. Notice that there is precisely one CompositeEntity.SeaSurface object instance for each ComponentEntity.NavigationSystem and that this plays the role of parent. The federate is implemented so that there is zero or one ComponentEntity.NavigationSystem object instance associated with the CompositeEntity.SeaSurface object instance that represents the parent entity. The federate will discover all CompositeEntity.SeaSurface objects within the federation but implements a filtering mechanism to reject attribute updates associated with instances that do not correspond to its parent entity.

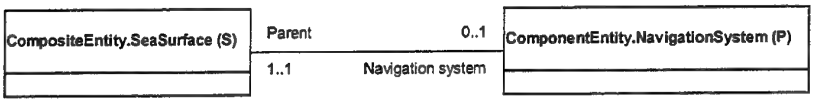


Figure 7.4a: FOM object collaboration diagram for the Helm federate.

The federate collaboration diagram for the Helm federate is illustrated in Figure 7.4b. This diagram illustrates the flow of data between the Helm federate and the Motion federate. The CompositeEntity.SeaSurface object instances that are published by the Motion federate are subscribed to by the Helm federate. It is through these subscriptions that the Helm federate is informed of the existence, state and kinematic attributes of its parent warship. As a minimum, the Helm federate will display the current position, speed and course of the warship.

The Helm federate collaborates with the VSEM federate through the mechanism of interactions, not object attribute updates. Interactions that are subclasses of the ExecutionManagement class[4] are sent from the VSEM to the Helm federate in order to manage the federation execution. The Helm federate sends ExecutionManagement.ExecutionManagementError interactions in the event that an instruction from the VSEM cannot be complied with.

The SystemControl.PropulsionSystemControl.SetPropulsionSystemAttribute interaction is sent to the Motion federate from the Helm federate to set the course and speed of an entity modelled by the Motion federate. Receipt of this interaction is acknowledged through the Motion federate sending the SystemControl.PropulsionSystemControl.AckSetPropulsionSystemAttribute interaction to the Helm federate.

[4] Excluding the ExecutionManagementError interaction that is sent from the Helm federate to the VSEM.

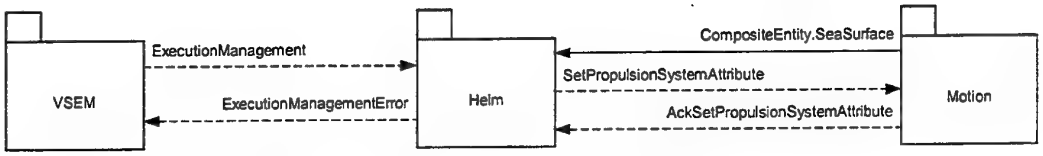


Figure 7.4b: Federate collaboration diagram for the Helm federate.

7.5 The Radar federate

The Radar federate simulates the performance of multiple radars. The principal inputs to the federate are the description and kinematic attributes of the composite entities that may be detected by the radar, or that may represent the parents of the radars. The principal outputs are track data.

The FOM object collaboration diagram is shown as Figure 7.5a. It illustrates that this federate publishes instances of the ComponentEntity.SensorSystem.RadarSystem object class to represent each of the radars it is modelling. Each radar performs zero or more tasks, and each of these is represented through publication of instances of the SensorTask.RadarTask object class. Each radar generates zero or more tracks and each of these is represented through publication of instances of the Track.RelativeTrack.RelativeRadarTrack object class. Hence, through publication of these object classes the Radar federate provides to the federation information concerning the existence of radars, the tasks they are performing and the track data generated by them as they detect and track targets.

This federate subscribes to the CompositeEntity.Air, CompositeEntity.SeaSurface and CompositeEntity.SubSurface object classes, and thereby receives data concerning the existence and kinematic attributes describing all composite entities within the scenario. Each of these entities may play one of two roles. They may either represent potential targets for the radar, or they may represent the parent composite entity that the radar is attached to. If it is the latter, the radar derives its location from that of its parent composite entity.

Figure 7.5a: FOM object collaboration diagram for the Radar federate.

The federate collaboration diagram for the Radar federate is illustrated in Figure 7.5b. This diagram illustrates the flow of data between the Radar federate and the Motion, Missile, Fusion, ESM and VSSD federates. The CompositeEntity.SeaSurface object instances that are published by the Motion federate are subscribed to by the Radar federate. It is through these subscriptions that the Radar federate is informed of the existence, state and kinematic attributes of its parent warship. In addition, the radar will detect other warships and the data required to compute its response is obtained through attributes of CompositeEntity.SeaSurface object instances. The CompositeEntity.Air object instance published by the Missile federate is subscribed to by the Radar federate. It is through this subscription that the Radar federate is informed of the existence, state and kinematic attributes of a potential target.

The Radar federate publishes instances of the ComponentEntity.SensorSystem.RadarSystem to represent radars, and SensorTask.RadarTask object instances to represent the tasks that the radars are performing. An ESM detects radars and therefore subscribes to these object instances in order to determine its response to them.

The Radar federate publishes instances of the Track.RelativeTrack.RelativeRadarTrack object class to provide track data to other federates. The Fusion federate subscribes to these object instances in order that it receive them and use them as inputs to its data fusion processes. In addition, the VSSD has the capacity to display track data and it therefore subscribes to the Track.RelativeTrack.RelativeRadarTrack object class. The VSSD has a feature that allows only those tracks to be displayed that are generated by a particular sensor. In order that it be able to identify the source of the track, the VSSD subscribes to the ComponentEntity.SensorSystem.RadarSystem object class.

The Radar federate collaborates with the VSEM federate through the mechanism of interactions, not object attribute updates. Interactions that are subclasses of the ExecutionManagement class[5] are sent from the VSEM to the Radar federate in order to manage the federation execution. The Radar federate sends ExecutionManagement.ExecutionManagementError interactions in the event that an instruction from the VSEM cannot be complied with.

[5] Excluding the ExecutionManagementError interaction that is sent from the Radar federate to the VSEM.

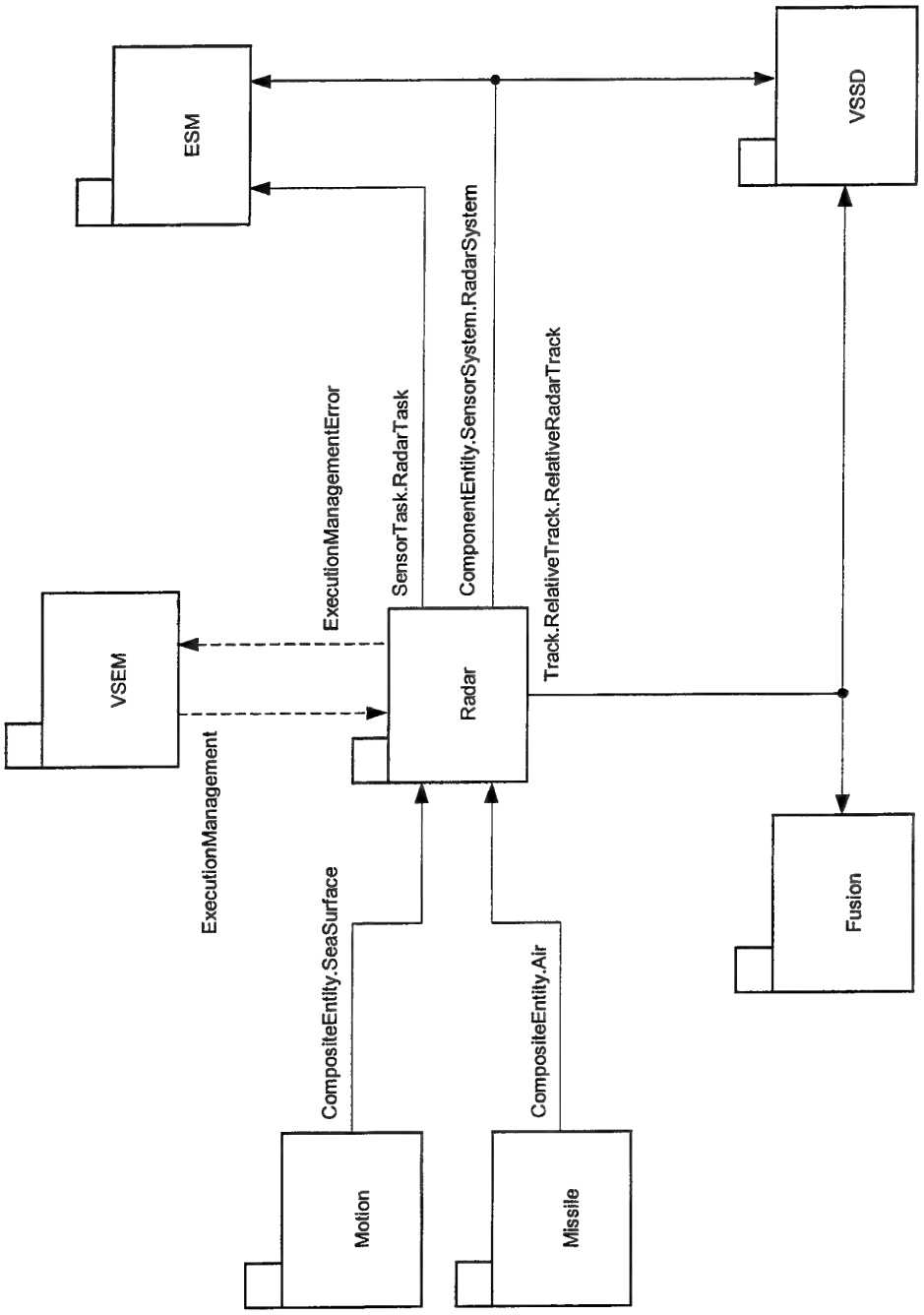


Figure 7.5b: Federate collaboration diagram for the Radar federate.

7.6 The IRST federate

The IRST federate simulates the performance of a single infrared search and track sensor. The principal inputs to the federate are the description and kinematic attributes of the composite entities that may be detected by the IRST and the principal outputs are track data.

The FOM object collaboration diagram is shown as Figure 7.6a. It illustrates that this federate publishes an instance of the ComponentEntity.SensorSystem.IRSystem to represent the IRST sensor it is modelling. The sensor performs zero or more tasks, and each of these is represented through publication of instances of the SensorTask.IRTask object class. The IRST generates zero or more tracks and each of these is represented through publication of instances of the Track.RelativeTrack.RelativeIRTrack object class. Hence, through publication of these object classes the IRST federate provides to the federation information concerning the existence of an IRST sensor, the tasks it is performing and the track data generated as it detects and tracks targets.

This federate subscribes to the CompositeEntity.Air, CompositeEntity.SeaSurface and CompositeEntity.SubSurface object classes, and thereby receives data concerning the existence and kinematic attributes describing all composite entities within the scenario. Each of these entities may play one of two roles. They may either represent potential targets for the IRST, or they may represent the parent composite entity that the IRST is attached to. If it is the latter, the IRST derives its location from that of its parent composite entity.

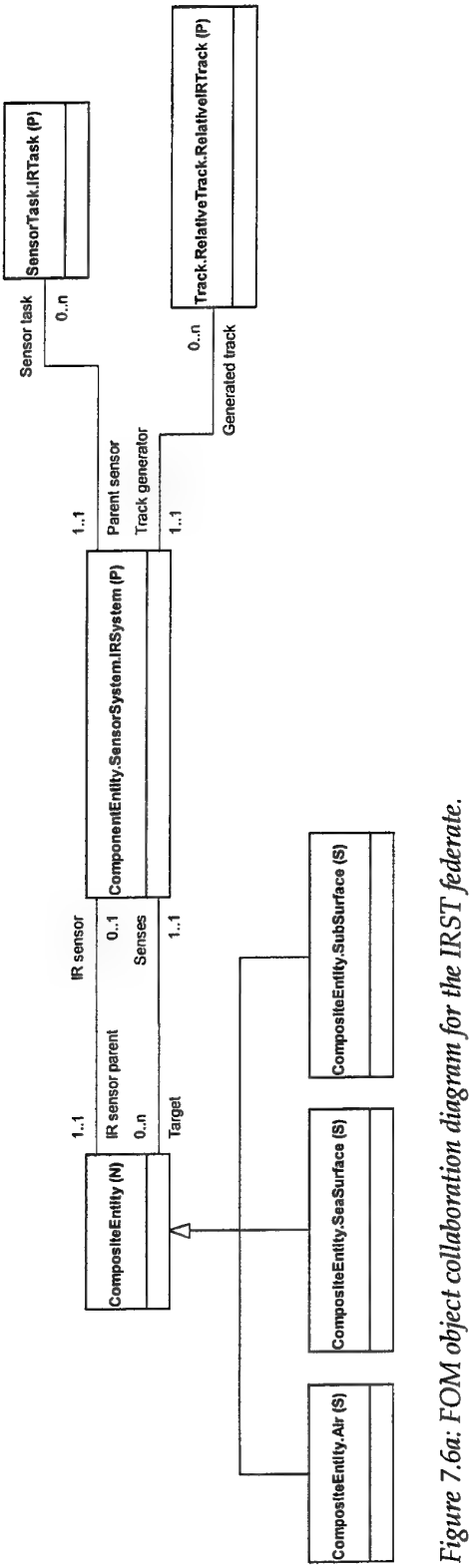


Figure 7.6a: FOM object collaboration diagram for the IRST federate.

The federate collaboration diagram for the IRST federate is illustrated in Figure 7.6b. This diagram illustrates the flow of data between the IRST federate and the Motion, Missile, Fusion and VSSD federates. The CompositeEntity.SeaSurface object instances that are published by the Motion federate are subscribed to by the IRST federate. It is through these subscriptions that the IRST federate is informed of the existence, state and kinematic attributes of its parent warship. In addition, the IRST will detect other warships and the data required to compute its response consists of the ships kinematic attributes and identity, obtained through attributes of CompositeEntity.SeaSurface object instances. The CompositeEntity.Air object instance published by the Missile federate is subscribed to by the IRST federate. It is through this subscription that the IRST federate is informed of the existence, state and kinematic attributes of a potential target.

The IRST federate publishes an instance of the ComponentEntity.SensorSystem.IRSystem to represent the IRST sensor it is modelling. Note that the FOM object collaboration diagram illustrates that this federate also publishes SensorTask.IRTask object instances to represent the tasks that the IRST sensor is performing. In this federation, though, no federate subscribes to this and it therefore does not appear in this diagram.

The IRST federate publishes instances of the Track.RelativeTrack.RelativeIRTrack object class to provide track data to other federates. The Fusion federate subscribes to these object instances in order that it receive them and use them as inputs to its data fusion processes. In addition, the VSSD has the capacity to display track data and it therefore subscribes to the Track.RelativeTrack.RelativeIRTrack object class. The VSSD has a feature that allows only those tracks to be displayed that are generated by a particular sensor. In order that it be able to identify the source of the track, the VSSD subscribes to the ComponentEntity.SensorSystem.IRSystem object class.

The IRST federate collaborates with the VSEM federate through the mechanism of interactions, not object attribute updates. Interactions that are subclasses of the ExecutionManagement class[6] are sent from the VSEM to the IRST federate in order to manage the federation execution. The IRST federate sends ExecutionManagement.ExecutionManagementError interactions in the event that an instruction from the VSEM cannot be complied with.

[6] Excluding the ExecutionManagementError interaction that is sent from the IRST federate to the VSEM.

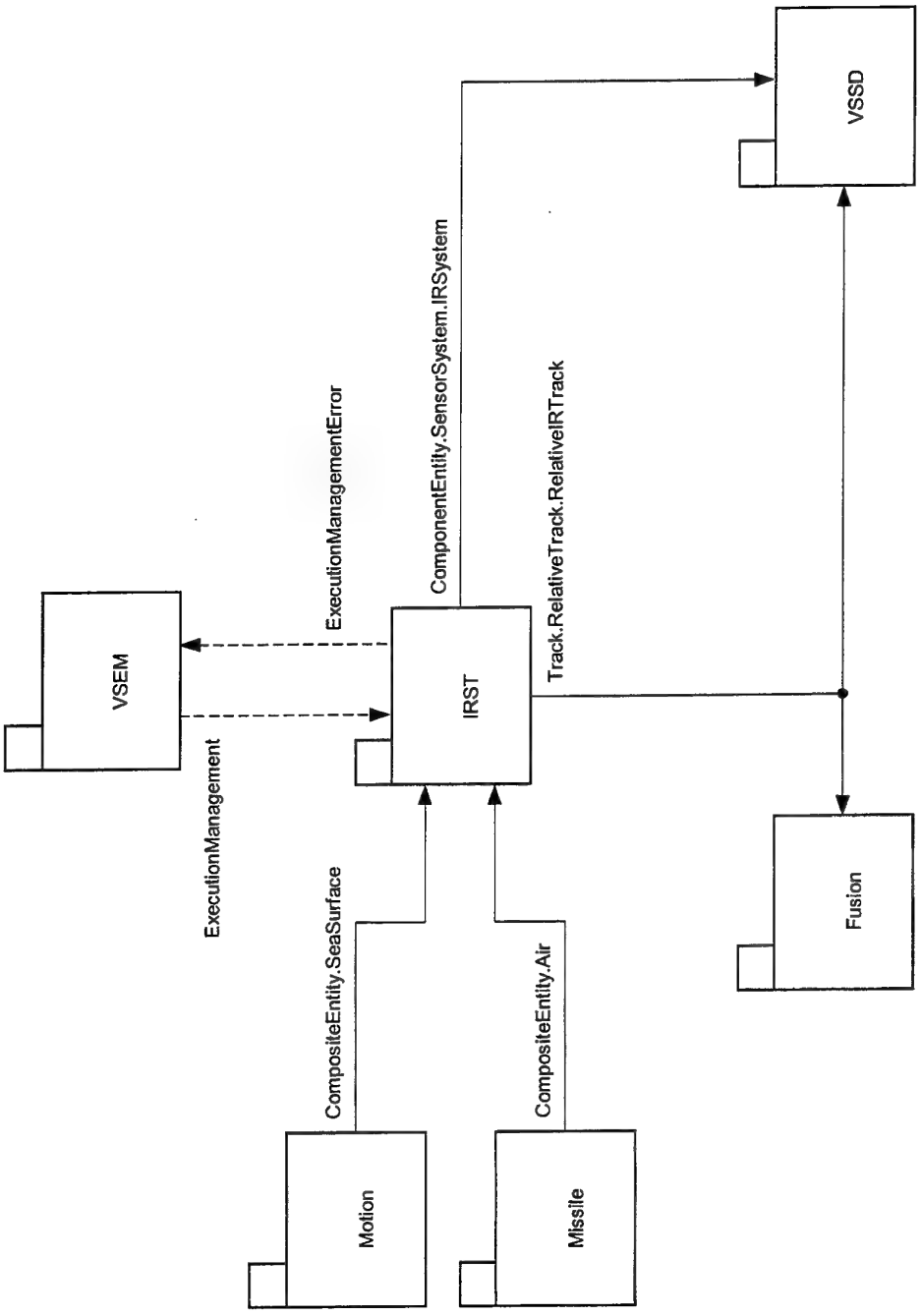


Figure 7.6b: Federate collaboration diagram for the IRST federate.

7.7 The ESM federate

The ESM federate simulates the performance of a single ESM sensor. The principal inputs to the federate are the description and kinematic attributes of the radars that may be detected by the ESM and the principal outputs are track data.

The FOM object collaboration diagram is shown as Figure 7.7a. It illustrates that this federate publishes an instance of the ComponentEntity.SensorSystem.ESMSystem to represent the ESM sensor it is modelling. The ESM performs zero or more tasks, and each of these is represented through the publication of instances of the SensorTask.ESMTask object class. The ESM generates zero or more tracks and each of these is represented through the publication of instances of the Track.RelativeTrack.RelativeESMTrack object class. Hence, through publication of these object classes the ESM federate provides to the federation information concerning the existence of an ESM sensor, the tasks it is performing and the track data generated by it as it detects and tracks emitting entities, specifically radars.

This federate subscribes to the ComponentEntity.SensorSystem.RadarSystem and SensorTask.RadarTask object instances. The targets for the ESM are radars, and these subscriptions provide to the federate the data it needs to determine its response to their transmissions. In addition, this federate subscribes to the CompositeEntity.Air, CompositeEntity.SeaSurface and CompositeEntity.SubSurface object classes, and thereby receives data concerning the existence and kinematic attributes of all composite entities within the scenario. Each of these entities may play one of two roles. They may either represent the parent entities of potential target radars, or they may represent the parent composite entity that the ESM is attached to. In the first case the ESM derives the location of target radars from that of their parent composite entities. In the second case the ESM derives its location from that of its parent composite entity. The ESM federate implements a filtering mechanism to reject as potential targets the radars with which it shares a parent entity.

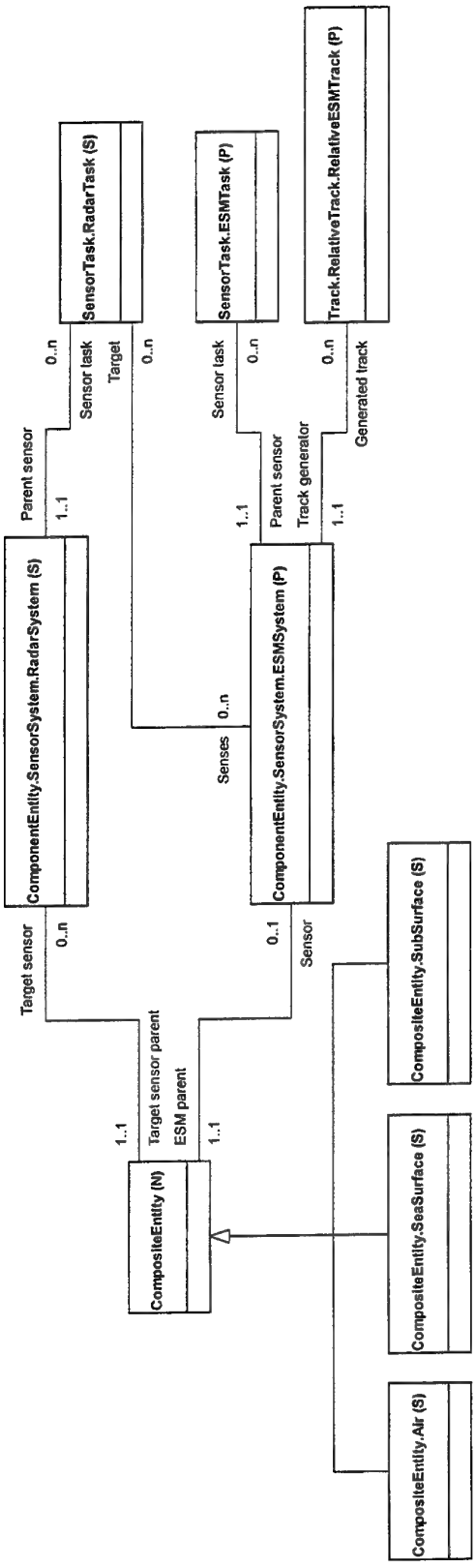


Figure 7.7a: FOM object collaboration diagram for the ESM federate.

The federate collaboration diagram for the ESM federate is illustrated in Figure 7.7b. This diagram illustrates the flow of data between the ESM federate and the Motion, Missile, Fusion, Radar and VSSD federates. The CompositeEntity.SeaSurface object instances that are published by the Motion federate are subscribed to by the ESM federate. It is through these subscriptions that the federate is informed of the existence, state and kinematic attributes of its parent warship. In addition, the ESM will discover other warships, which may be the parents of target radars. The CompositeEntity.Air object instance published by the Missile federate is subscribed to by the ESM federate. It is through this subscription that the ESM federate is informed of the existence, state and kinematic attributes of the parent of the missile seeker, which is a potential target.

The targets for the ESM are radars, hence it subscribes to the ComponentEntity.SensorSystem.RadarSystem and SensorTask.RadarTask object classes. These are published by the radar federate and the missile federate, noting that the missile seeker is a radar, which is modelled within this federate.

The ESM federate publishes an instance of the ComponentEntity.SensorSystem.ESMSystem to represent the ESM sensor. Note that the FOM object collaboration diagram illustrates that this federate also publishes SensorTask.ESMTask object instances to represent the tasks that the ESM is performing. In this federation, though, no federate subscribes to this and it therefore does not appear in this diagram.

The ESM federate publishes instances of the Track.RelativeTrack.RelativeESMTrack object class to provide track data to other federates. The Fusion federate subscribes to these object instances in order that it receive them and use them as inputs to its data fusion processes. In addition, the VSSD has the capacity to display track data and it therefore subscribes to the Track.RelativeTrack.RelativeESMTrack object class. The VSSD has a feature that allows only those tracks to be displayed that are generated by a particular sensor. In order that it be able to identify the source of the track, the VSSD subscribes to the ComponentEntity.SensorSystem.ESMSystem object class.

The ESM federate collaborates with the VSEM federate through the mechanism of interactions, not object attribute updates. Interactions that are subclasses of the ExecutionManagement class[7] are sent from the VSEM to the ESM federate in order to manage the federation execution. The ESM federate sends ExecutionManagement.ExecutionManagementError interactions in the event that an instruction from the VSEM cannot be complied with.

[7] Excluding the ExecutionManagementError interaction that is sent from the ESM federate to the VSEM.

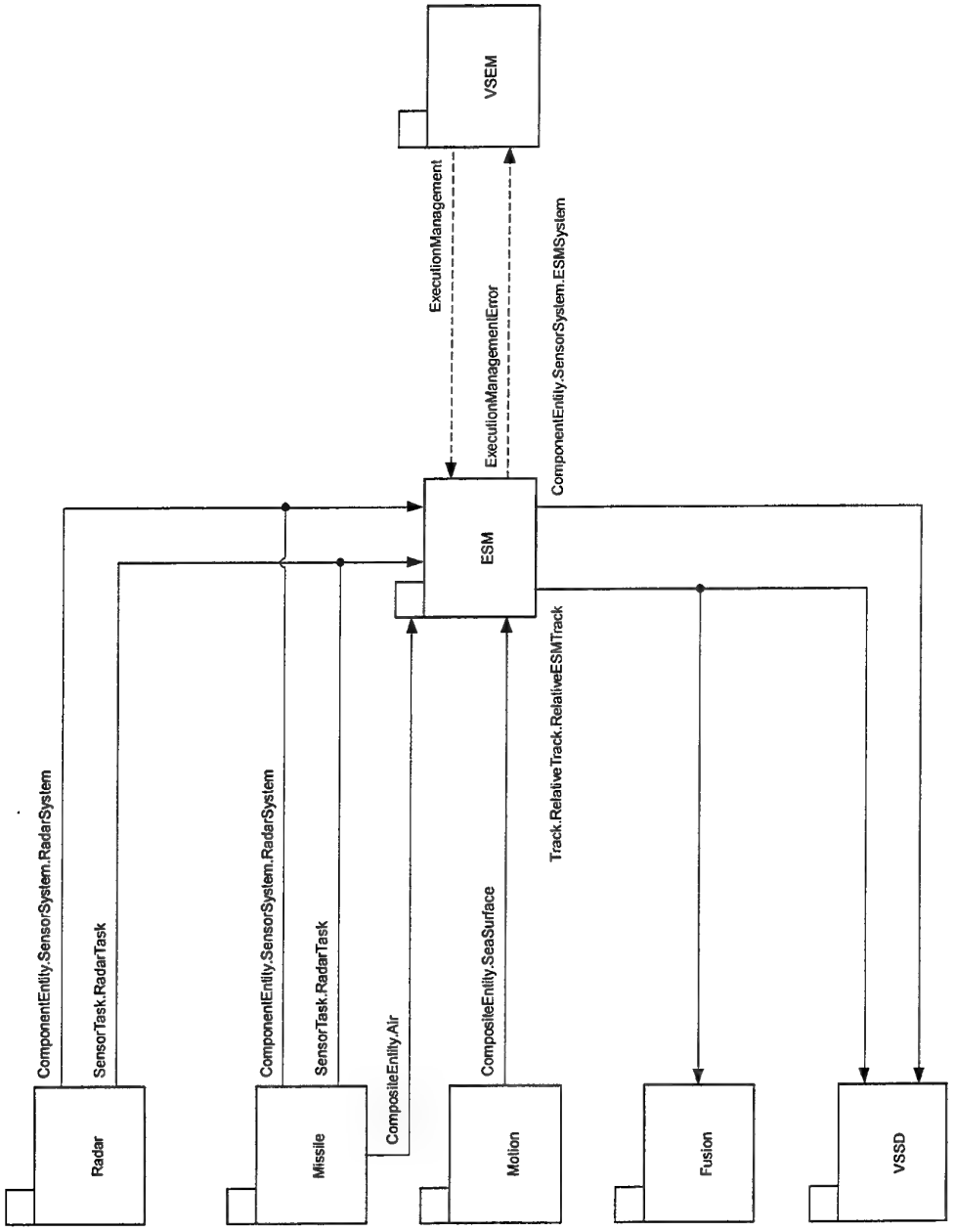


Figure 7.7b: Federate collaboration diagram for the ESM federate.

7.8 The Fusion federate

The Fusion federate fuses the track data from the warship's sensors to create a tactical picture. The principal inputs to the federate are track data from sensor federates and the outputs are fused tracks.

The FOM object collaboration diagram is shown as Figure 7.8a. It illustrates that this federate publishes an instance of the ComponentEntity.CommandAndControlSystem to represent the fusion system. The fusion system generates zero or more fused tracks and each of these is represented through publication of instances of the Track.RelativeTrack.RelativeFusedTrack and Track.AbsoluteTrack.AbsoluteFusedTrack object classes. Hence, through publication of these object classes the Fusion federate provides to the federation information concerning the existence of the fusion system and the track data generated by it.

This federate subscribes to the Track.RelativeTrack.RelativeIRTrack, Track.RelativeTrack.RelativeESMTrack and Track.RelativeTrack.RelativeRadarTrack object classes and thereby receives tracks generated by the sensors of the warship. A key input to the fusion algorithm is the location of the entity on which the fusion system resides. Hence it subscribes to the CompositeEntity.Air, CompositeEntity.SeaSurface and CompositeEntity.SubSurface object classes. Although in this scenario the parent of the fusion system is a warship, it may be considered as attached to any composite entity. The federate implements a filtering mechanism to ignore those composite entities it discovers that are not its parent.

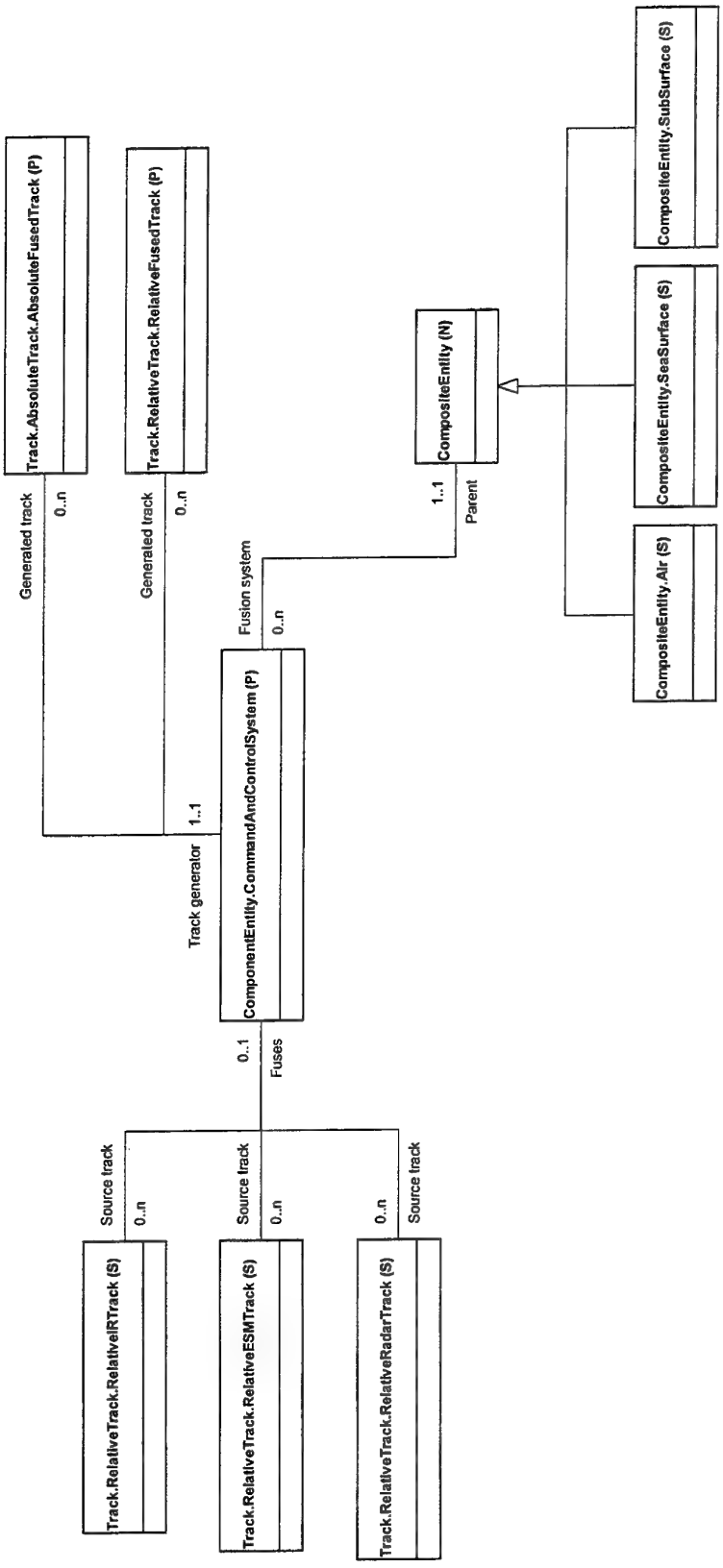


Figure 7.8a: FOM object collaboration diagram for the Fusion federate.

The federate collaboration diagram for the Fusion federate is illustrated in Figure 7.8b. This diagram illustrates the flow of data between the Fusion federate and the Motion, ESM, Radar, IRST and VSSD federates. The CompositeEntity.SeaSurface object instances published by the Motion federate are subscribed to by the Fusion federate. It is through these subscriptions that the Fusion federate is informed of the existence, state and kinematic attributes of its parent warship, and these data may be required as inputs to fusion algorithms. The federate subscribe to the Track.RelativeTrack.RelativeRadarTrack, Track.RelativeTrack.RelativeESMTrack and Track.RelativeTrack.RelativeIRTrack object classes that are respectively published by the Radar, ESM and IRST federates. These data are the principal inputs to the fusion algorithm.

The Fusion federate publishes instances of the ComponentEntity.CommandAndControlSystem to represent the fusion system, and instances of the Track.RelativeTrack.RelativeFusedTrack and Track.AbsoluteTrack.AbsoluteFusedTrack object classes to provide track data to other federates. The VSSD has the capacity to display track data and it therefore subscribes to the Track.RelativeTrack.RelativeFusedTrack and Track.AbsoluteTrack.AbsoluteFusedTrack object classes. The VSSD has a feature that allows only those tracks to be displayed that are generated by a particular sensor. In order that it be able to identify the source of the track, the VSSD subscribes to the ComponentEntity.CommandAndControlSystem object class.

The Fusion federate collaborates with the VSEM federate through the mechanism of interactions, not object attribute updates. Interactions that are subclasses of the ExecutionManagement class[8] are sent from the VSEM to the Fusion federate in order to manage the federation execution. The Fusion federate sends ExecutionManagement.ExecutionManagementError interactions in the event that an instruction from the VSEM cannot be complied with.

[8] Excluding the ExecutionManagementError interaction that is sent from the Fusion federate to the VSEM.

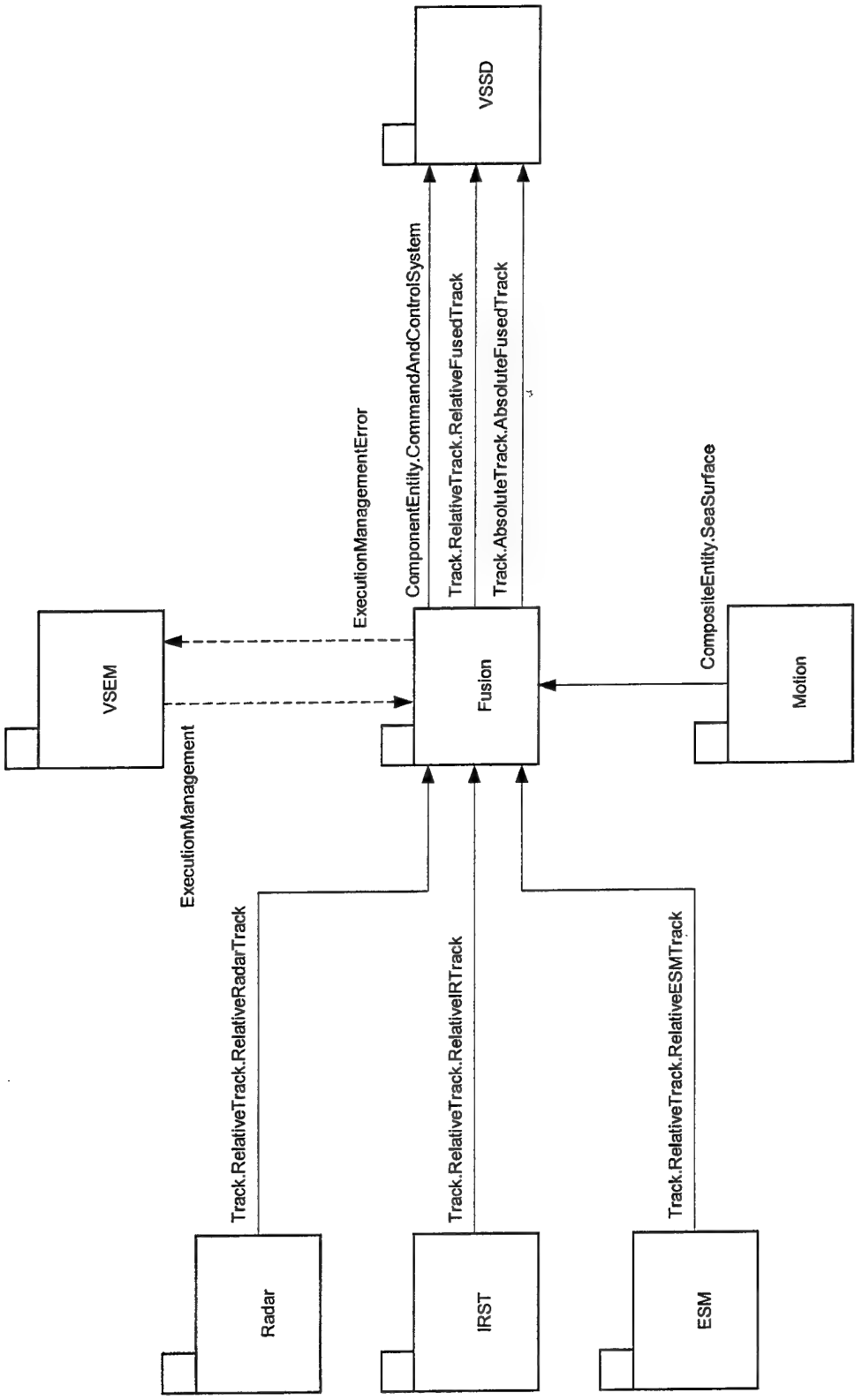


Figure 7.8b: Federate collaboration diagram for the Fusion federate.

7.9 The Missile federate

The Missile federate simulates the behaviour of an anti-ship missile. In doing so it simulates not only the flight of the missile but the performance of its seeker, which in this case is a radar. The principal inputs to the federate are the description and kinematic attributes of the composite entities that may be detected by its seeker and the principal outputs are the description and kinematic attributes of the missile, along with a description of the behaviour of the seeker.

The FOM object collaboration diagram is shown as Figure 7.9a. It illustrates that this federate publishes an instance of the CompositeEntity.Air object class to represent the missile, and an instance of the ComponentEntity.SensorSystem.RadarSystem to represent the missile seeker. The seeker performs zero or more tasks, and each of these is represented through publication of instances of the SensorTask.RadarTask object class. Hence, through publication of these object classes the Missile federate provides to the federation information concerning the existence of the missile, its seeker and the tasks the seeker is performing.

This federate subscribes to the CompositeEntity.Air, CompositeEntity.SeaSurface and CompositeEntity.SubSurface object classes, and thereby receives data concerning the existence and kinematic attributes describing all composite entities within the scenario. Each of these entities may represent potential targets for the missile.

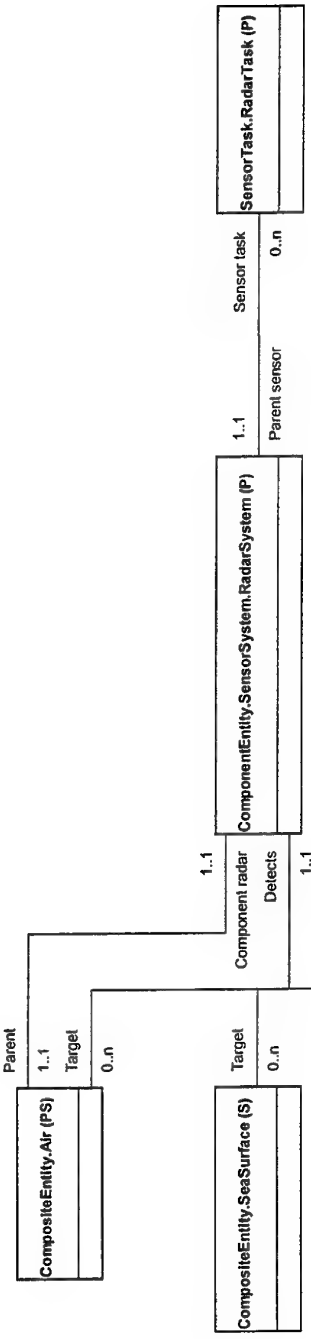


Figure 7.9a: FOM object collaboration diagram for the Missile federate.

The federate collaboration diagram for the Missile federate is illustrated in Figure 7.9b. This diagram illustrates the flow of data between the Missile federate and the Motion, Radar, ESM, IRST and VSSD federates. The CompositeEntity.SeaSurface object instances that are published by the Motion federate are subscribed to by the Missile federate. It is through these subscriptions that the Missile federate is informed of the existence, state and kinematic attributes of potential targets.

The Missile federate publishes an instance of the CompositeEntity.Air object class to represent the missile. This object class is subscribed to by the Radar, IRST, ESM and VSSD federates. The Radar and IRST federates use these data to determine their response to the potential target. The VSSD uses these data to display missile entities. The ESM detects the radar seeker, so it additionally subscribes to the ComponentEntity.SensorSystem.RadarSystem and SensorTask.RadarTask object instances published by the Missile federates. The location of this radar is derived from the attributes of the CompositeEntity.Air object instance.

The VSSD discovers the instance of the ComponentEntity.SensorSystem.RadarSystem object class published by the Missile federate. The VSSD attempts to associate track data with this object, but fails given that the Missile federate does not publish track data.

The Missile federate collaborates with the VSEM federate through the mechanism of interactions, not object attribute updates. Interactions that are subclasses of the ExecutionManagement class[9] are sent from the VSEM to the Missile federate in order to manage the federation execution. The Missile federate sends ExecutionManagement.ExecutionManagementError interactions in the event that an instruction from the VSEM cannot be complied with.

[9] Excluding the ExecutionManagementError interaction that is sent from the Missile federate to the VSEM.

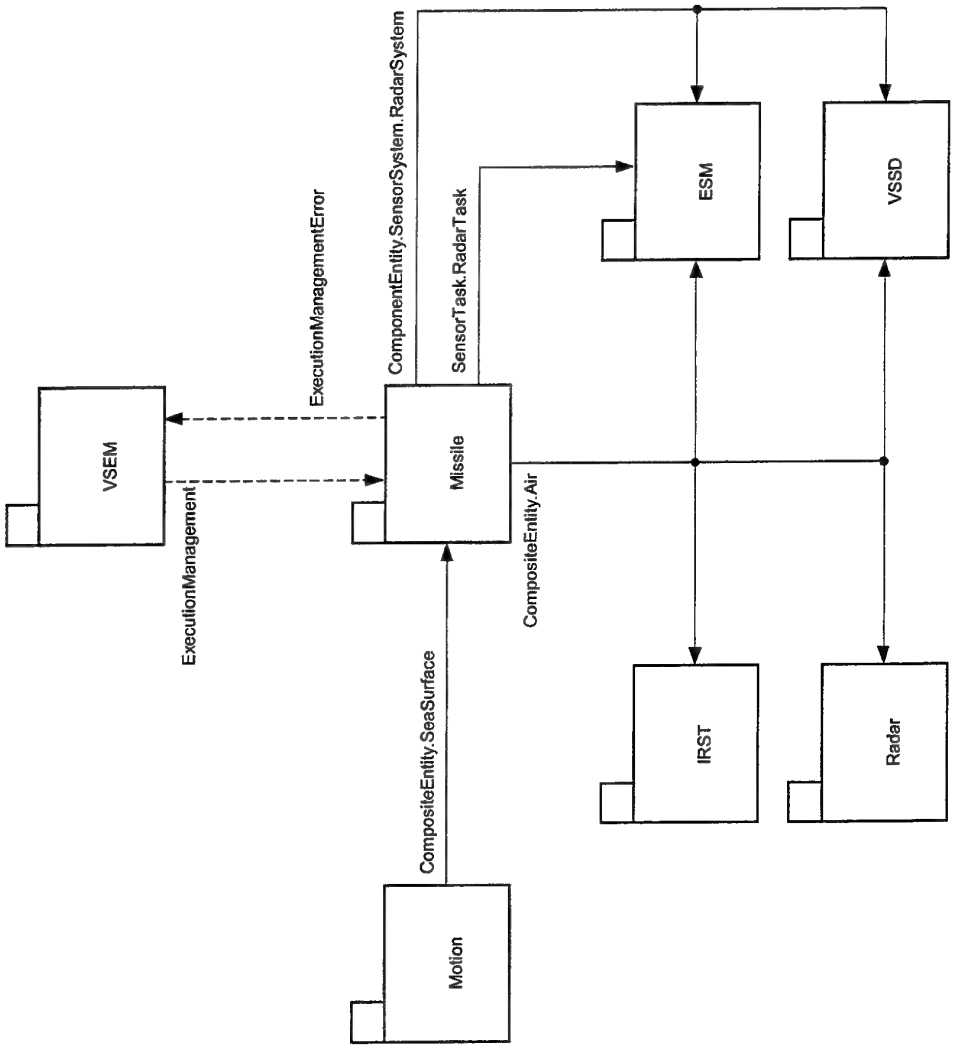


Figure 7.9b: Federate collaboration diagram for the Missile federate.

7.10 The Virtual Ship Simulation Display (VSSD) federate

The VSSD federate provides a plan view display of the federation in addition to displaying the tracks generated by the sensors and fusion system. The principal inputs to the federate are the description and kinematic attributes of the composite entities that exist within the scenario and the track data generated by the Radar, ESM, IRST and Fusion federates. There are no outputs from this federate.

The FOM object collaboration diagram is shown as Figure 7.10a. It illustrates that this federate subscribes to the CompositeEntity.Air, CompositeEntity.SeaSurface and CompositeEntity.SubSurface object classes. Through discovery of instances of these, the federates receives the data required to generate a plan view display of the composite entities within the federation.

In order to display the track data generated within the federation the VSSD federate subscribes to the Track.RelativeTrack.RelativeFusedTrack, Track.AbsoluteTrack.AbsoluteFusedTrack, Track.RelativeTrack.RelativeESMTrack, Track.RelativeTrack.RelativeIRTrack and Track.RelativeTrack.RelativeRadarTrack object classes. In order to identify the system that generates each track the VSSD federate also subscribes to the ComponentEntity.SensorSystem.ESMSystem, ComponentEntity.SensorSystem.IRSystem, ComponentEntity.SensorSystem.RadarSystem and ComponentEntity.CommandAndControlSystem object classes.

Note also that the CompositeEntity.Air, CompositeEntity.SeaSurface and CompositeEntity.SubSurface object classes may play a role additional to representing actual entities for display. They also play the role of parents of sensors, information that may be exploited in selectively displaying tracks generated by the sensors of a single composite entity.

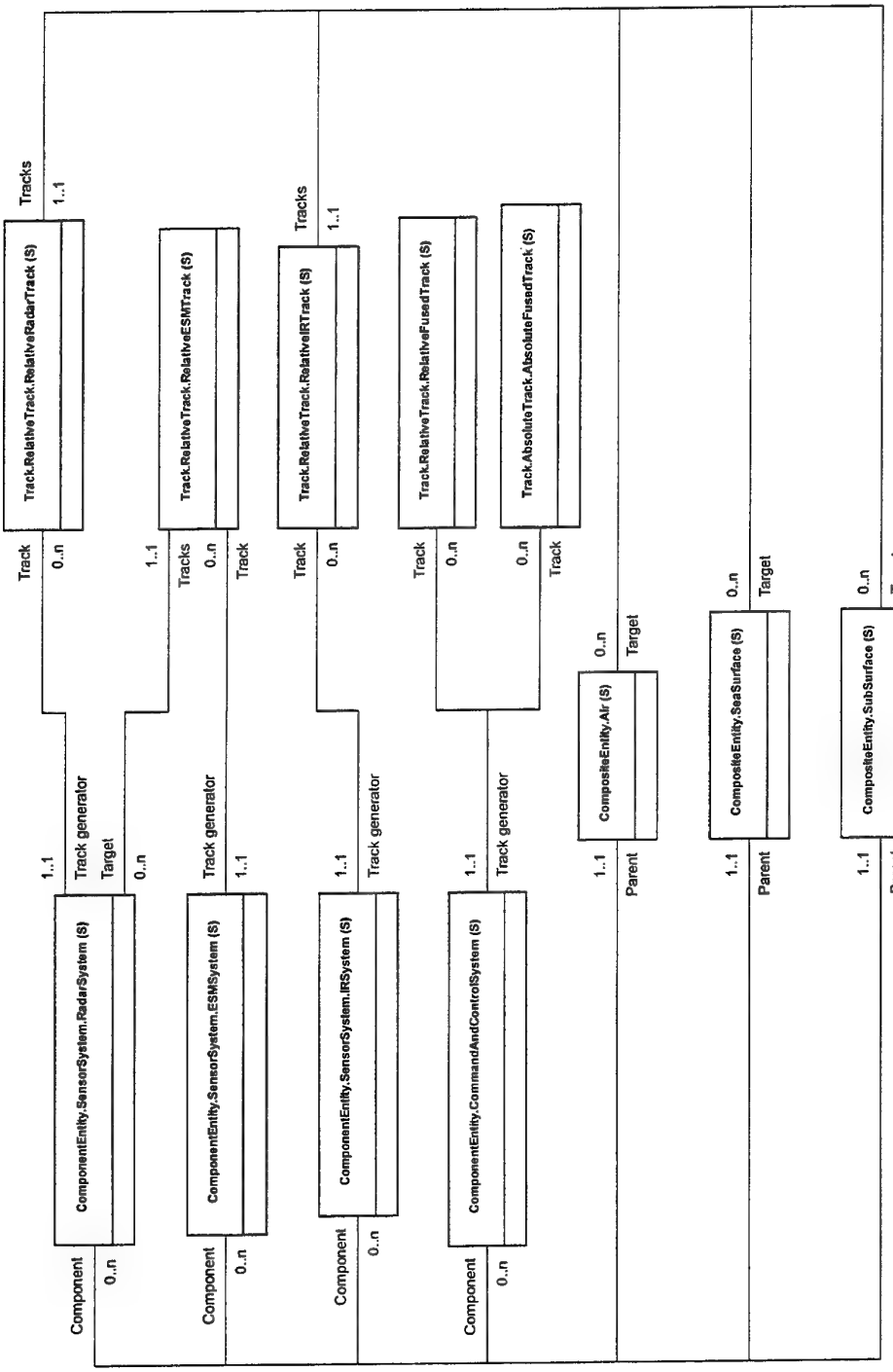


Figure 7.10a: FOM object collaboration diagram for the VSSD federate.

The federate collaboration diagram for the VSSD federate is illustrated in Figure 7.10b. This diagram illustrates the flow of data between the VSSD federate and the Motion, Missile, Fusion, ESM, IRST and Radar federates. The CompositeEntity.SeaSurface object instances that are published by the Motion federate are subscribed to by the VSSD federate. It is through these subscriptions that the VSSD federate is informed of the existence, state and kinematic attributes of warships and can therefore display them. These object instances may additionally play the role of the parent of track generating systems; in this case Radar, ESM, IRST and Fusion systems. In addition, instances of the CompositeEntity.Air object class published by the Missile federate are discovered and displayed.

The Track.RelativeTrack.RelativeRadarTrack, Track.RelativeTrack. RelativeESMTrack, Track.RelativeTrack. RelativeIRTrack, Track.RelativeTrack. RelativeFusedTrack and Track.AbsoluteTrack.AbsoluteFusedTrack object classes are subscribed by this federate in order that they may be displayed. In addition, the ComponentEntity.SensorSystem.RadarSystem, ComponentEntity.SensorSystem.ESMSystem, ComponentEntity.SensorSystem.IRSystem and ComponentEntity.CommandAndControlSystem object classes are subscribed in order that the source of a track can be identified. Tracks may thus be displayed according to the system or composite entity from which they originate.

The VSSD federate collaborates with the VSEM federate through the mechanism of interactions, not object attribute updates. Interactions that are subclasses of the ExecutionManagement class[10] are sent from the VSEM to the VSSD federate in order to manage the federation execution. The VSSD federate sends ExecutionManagement.ExecutionManagementError interactions in the event that an instruction from the VSEM cannot be complied with.

[10] Excluding the ExecutionManagementError interaction that is sent from the VSSD federate to the VSEM.

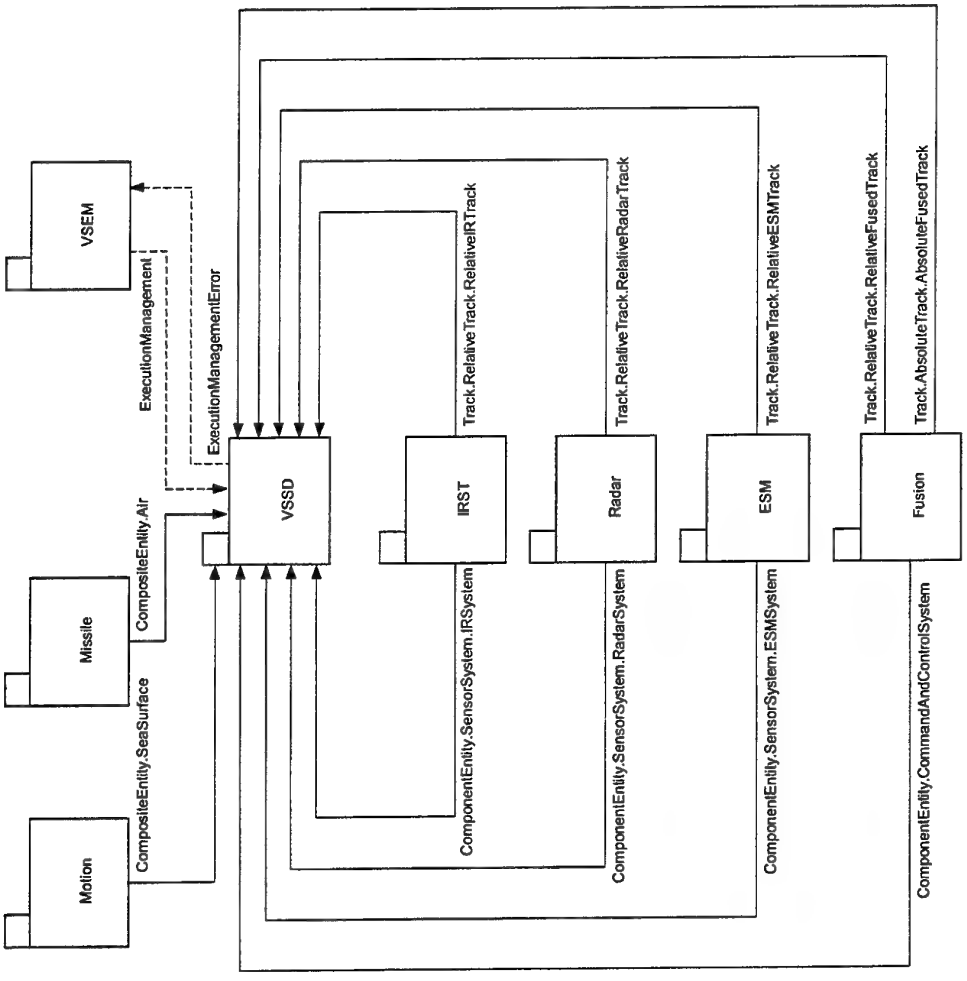


Figure 7.10b Federate collaboration diagram for the VSSD federate.

8. Execution management

A requirement for any distributed system is coordination of the activities of the components. In the context of a HLA federation, coordination of the federates is required, and the RTI provides a basic set of services to facilitate this. For example, the federation management services provide the means to create and destroy federations and by which individual federates join and resign from a federation execution. The time management services provide for coordination of the time advances of federates. However, it is evident that there is a requirement to impose a level of coordination above that provided by the RTI services.

In the first instance, although federates may require data from other federates, they are typically designed to continue to operate in the absence of such data and there is no RTI service that provides advice that required data sources are absent. The essential problem is one of ensuring that all federates required for a federation execution have joined before the simulation proceeds. It is also typically required to have a mechanism whereby a specified initial state for the federation can be defined and an assurance obtained that the specified initial state is established before the simulation proceeds.

These two control features are essential in the event that fully repeatable federation executions are to take place. There are two requirements for repeatability; sameness of initial conditions and sameness of inputs throughout the simulation. The requirements outlined in the previous paragraph contribute to ensuring that the initial conditions of the federation are identical and replicable. The use of the time management services provides the other component of repeatability under conditions on identical inputs throughout the federation execution.

It should be noted that the Virtual Ship may be used to support real time simulation in which time management services are not used. Even under such circumstances the conditions detailed previously are required to ensure that the federates interact in a meaningful fashion.

The requirement for the Virtual Ship Architecture calls for support of scenario replication in order to enable Monte Carlo techniques. Because of the intrinsic distributed nature of a Virtual Ship federation and the difficulty and time required to remotely activate federates, the solution has been to provide for replication of scenario execution within a single federation execution. This involves deleting object instances at the conclusion of a scenario execution followed by recreation of the scenario initial conditions. It is particularly critical to reset the federation time to zero.

Support for multiple scenario executions within a single federation execution requires that a mechanism be put in place whereby each replication can be uniquely identified. This facilitates the identification of data sets, and other simulation outputs, with a particular scenario.

The use of Monte Carlo techniques also typically involves the use of random variations of conditions from one scenario to another. Hence it is required to generate and promulgate random number seeds throughout the federation.

It is these services that are not provided by the RTI but are considered necessary to enable a Virtual Ship federation execution to occur in a controlled way and a repeatable way, if desired. It is the provision of these services that we refer to as execution management.

This section outlines the Virtual Ship concept for execution management and it is described in detail in [2]. The concept essentially defines the high level flow of a Virtual Ship federation execution and this is described in terms of the federation undergoing transitions amongst a finite collection of states. To implement the concept requires three components: a federate known as the Virtual Ship Execution Manager (VSEM), a file (known as the VSEM script file) input to the VSEM that details how the federation execution should take place and the functionality built into the other federates to interact with the VSEM.

The flow of a managed federation is shown in Figure 8a and described as follows. The initial phase consists of all federates joining the federation. The VSEM prevents the federation proceeding until all federates have joined. The next phase is an initialisation phase, during which federates perform internal initialisations, such as establishing their time management policies. The next phase is a federation save. Each of the federates saves its initial state, which should be null in the sense that no entities exist. In addition, the initial time of the federation is saved. It is the use of the save functionality that enables multiple replications of a scenario within a single federation execution.

Following this phase is a restore phase, where the initial state of the federation is restored and the federation time set to its initial value. Having restored the federation the VSEM may provide to the federates a descriptor and random number seed for this replication of the scenario. The federation now enters the simulate phase and all federates behave according to the entities they are representing. During this phase, the VSEM may send messages to the federates advising them to create composite and component entities. These entities may be created at any time during the scenario, but those created at initial time constitute the initial conditions of the federation.

It is this process that provides an assurance that the entities required within a scenario have been created. These messages are sent in time stamp order (TSO) and all federates that are time constrained will receive these in accordance with the time management algorithms. If the federates that receive these messages are time regulating they will respond to entity creation messages by registering object instances and providing an initial attribute update, which is sent TSO. Hence, a federation in which all federates are time regulating and time constrained cannot advance until the federates have received these TSO messages describing the initial entities within the federation.

At the conclusion of the scenario, which is determined on the basis of elapsed time, a scenario shutdown phase occurs, when each federate may write scenario specific data to file and perform any other task required for a graceful shutdown of a scenario.

If the scenario is to be repeated the restore phase in executed next, and the process repeats as many times as requested. When the final iteration of the scenario is complete, a federation shutdown phase occurs, during which all federates resign from the federation and it is destroyed.

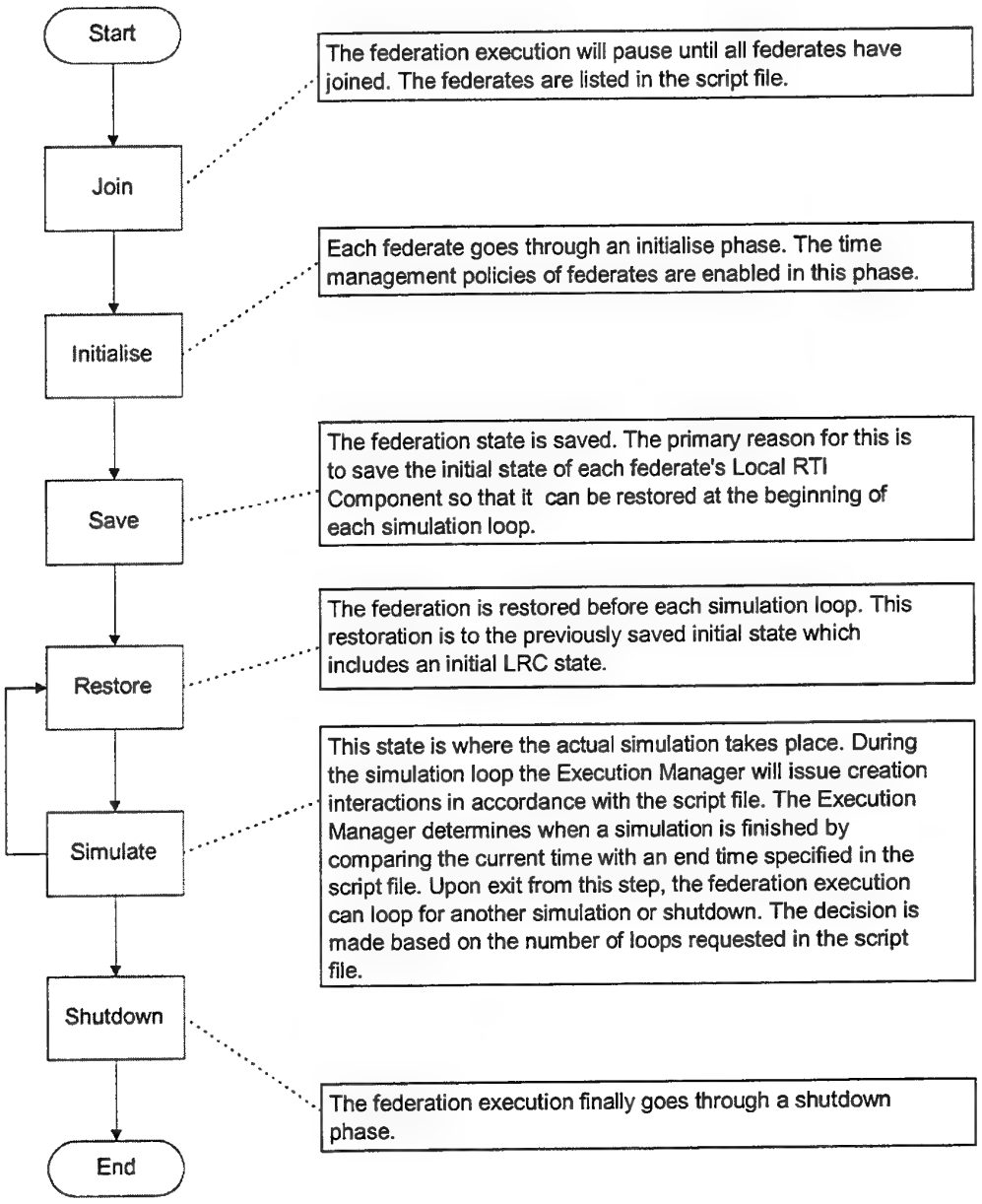


Figure 8a: Flow of an execution managed federation execution.

The VSEM script file provides the specific details of this process. In particular, it lists the federates that make up the federation, the number of loops of the scenario that are to be executed, whether scenario descriptions and random number seeds are required and the list of composite and component entities that are to be created during the scenario, along with the time at which they are created. The scenario end time is also specified.

As detailed above, the federates receive and respond to messages from the VSEM. The VSEM additionally makes use of synchronisation points to control the flow of execution through the various phases. Hence, federates that interact with the VSEM are required to be constructed to respond to the interactions sent by it, and to respond appropriately to the synchronisation points. These design requirements are specified in detail in *Execution Management in the Virtual Ship Architecture* [2].

9. Issues associated with modelling sensors

Modelling sensor performance is at the heart of simulation in the maritime domain. From the perspective of a warship, situation awareness is almost exclusively acquired through sensors[11]. In addition, advanced weapon systems make extensive use of sensors to facilitate targeting and weapon guidance. Hence, modelling signal propagation is of paramount importance.

A critical requirement to model signal propagation is a representation of the environment. Representation of the environment, however, extends beyond the issue of propagation modelling and is addressed in section 10.

9.1 Treatment of propagation

Within the Virtual Ship Architecture propagation is modelled within federates that model sensors. Using the federate taxonomy introduced in section 5, these are the sensor system and external entity federates. Although not specifically acknowledged in section 5, variations of the countermeasure system and weapon system federates may model sensors within them and hence be required to model propagation.

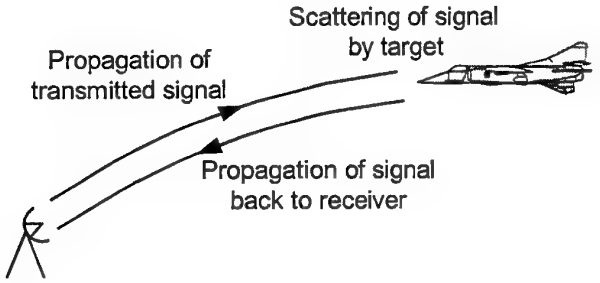


Figure 9.1a: A federate representing an active sensor, such as a radar, models propagation of its transmitted signal to a target, the scattering behaviour of the target and propagation of the scattered signal back to the receiver.

To elaborate by way of example, a federate representing a radar models propagation of a signal from the radar to a potential target, the scattering of the signal by the target and propagation back to the radar. In the example of an ESM, or a passive sonar, the federate representing these sensors models the emissions of potential targets and the propagation of these signals to the sensor. These examples are illustrated respectively in Figures 9.1a and 9.1b.

There are a number of reasons that drive this approach. In the first instance, the domain expertise to model propagation typically resides with the sensor expert. Such an expert similarly possesses the expertise to model scattering behaviour in the case of active sensors and emitting behaviour in the case of passive sensors.

As a result of adopting this approach, a federate modelling a sensor must have a representation of all entities that may be sensed by it, to the degree of fidelity

[11] Information concerning the situation can be obtained from third parties, for example via tactical data links, but the data so obtained is, in all likelihood, obtained from various sensors.

required to perform the modelling. Taking the example of a radar federate, for every potential target the radar federate requires a representation of its scattering behaviour, whether it be represented simply in the form of a single radar cross section value, or in a more complicated manner, as a function of incident and scattered directions. An association is required between the identity of the target and the representation of the scattering behaviour. The Name attribute of the CompositeEntity object instance is used to facilitate this.

An example algorithm for computing the signal received by an active sensor is as follows:

1. Reflect the Name, Position, Velocity, Acceleration, Orientation, OrientationRate and DRAlgorithm attributes of the CompositeEntity object instance that represents the target.
2. Compute the current position of the target, using the Position, Velocity, Acceleration, Orientation, OrientationRate attributes and dead reckoning formulae indicated by the DRAlgorithm attribute value.
3. Compute propagation of the signal from transmitter to target using knowledge of own position and knowledge of target position.
4. Look up scattering properties of the target using Name attribute.
5. Compute the signal scattered by the target.
6. Compute propagation of the signal from target to receiver.

An algorithm for the passive case is presented in the next section.

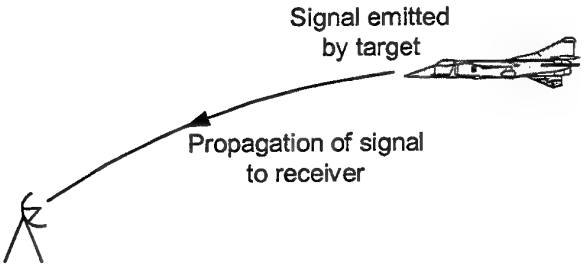


Figure 9.1b: A federate representing a passive sensor, such as an ESM, models the emission of signals by the target and propagation of these signals to the receiver.

This approach to treating propagation is typically straightforward to implement. The alternative solution is to provide a central federate within which all propagation is computed. In such an approach, a sensor federate would provide to the propagation federate a description of the transmitted signal. The propagation federate would then provide to the receiving federate a description of the received signal. In the case where the receiving federate modelled a scattering entity, this federate would provide to the propagation federate a description of the scattered signal. The propagation federate would then compute the propagation of this signal to receiving federates that would determine their response upon its receipt. Proposals for such an implementation have been provided by Best & Luckman [5] and Cramp [6]. Some difficulties inherent in this approach are scalability and the challenge of representing complex signals in an economical way. The approach adopted here reflects the lack of clarity in how to proceed to implement a central propagation federate. However,

implementing such a concept is an objective for future iterations of the VSA (see section 15).

9.2 Passive detection of transient signals

A significant challenge is faced in modelling the receipt of transient signals by passive detectors. The archetypal problem is that of an ESM detecting the transmissions of a radar. In accordance with the assumption that the ESM federate models propagation from the radar to the ESM, this federate requires a description of the signals transmitted, the time at which they are transmitted and information concerning the direction in which the transmission occurs. Information concerning the transmitting beam pattern is assumed implicit in the description of the signal transmitted.

Carrying forward the example of the ESM, it is assumed that this federate determines the nature of the transmitted signal on the basis of the identity of the emitting system. Just as the radar federate will associate scattering properties with an entity using the Name attribute of the CompositeEntity instance, so will an ESM federate associate transmitting behaviour with a system entity using the ComponentName attribute of the ComponentEntity object instance (see section 12).

In this regard, the approach to modelling detection of transient signals is completely analogous to modelling active sensors; the federate representing the sensor is assumed to possess knowledge of the emissions of the detected entity. The case of passive detection differs in that additional information is required concerning the directionality of emitted signals. This information is conveyed by the attributes of the SensorTask object instance. A coordinate system is assumed attached to the beam pattern describing the emissions and this is illustrated in Figure 9.2b. The BeamPatternOrientation and BeamPatternOrientationRate attributes allow the motion of the beam pattern in time to be determined (see *Coordinate Usage in the Virtual Ship Architecture* [1]). With this information, a federate representing a passive sensor can compute propagation from the emitter to the detector. In addition, side lobes can be modelled, with the structure of the beam pattern assumed known by the sensor federate and associated with an emitting entity via the ComponentName attribute.

An example algorithm for computing the signal received by a passive sensor is as follows:

1. Reflect the Position, Velocity, Acceleration, Orientation, OrientationRate and DRAlgorithm attributes of the CompositeEntity object instance (the parent entity) to which the emitting entity is attached.
2. Reflect the ComponentName, RelativePosition and RelativeOrientation attributes of the ComponentEntity object instance that represents the emitting entity.
3. Reflect the BeamPatternReference, BeamPatternOrientation, BeamPatternOrientationRate and BeamPatternDRAlgorithm attributes of the SensorTask object instance that represents a set of emissions (task) by the emitting entity.
4. Compute the current position of the target, using the Position, Velocity, Acceleration, Orientation, OrientationRate and DRAlgorithm attributes of the

parent CompositeEntity, and the RelativePosition and RelativeOrientation attributes of the ComponentEntity.

5. Look up the emitting properties of the emitter using the ComponentName attribute.
6. Compute the current orientation of the beam pattern using the BeamPatternReference, BeamPatternOrientation, BeamPatternOrientationRate and BeamPatternDRAlgorithm attributes.
7. Compute the signal emitted towards the receiver using the current orientation and properties of the beam pattern.
8. Compute propagation of the signal from emitter to receiver using knowledge of own position and knowledge of target position.

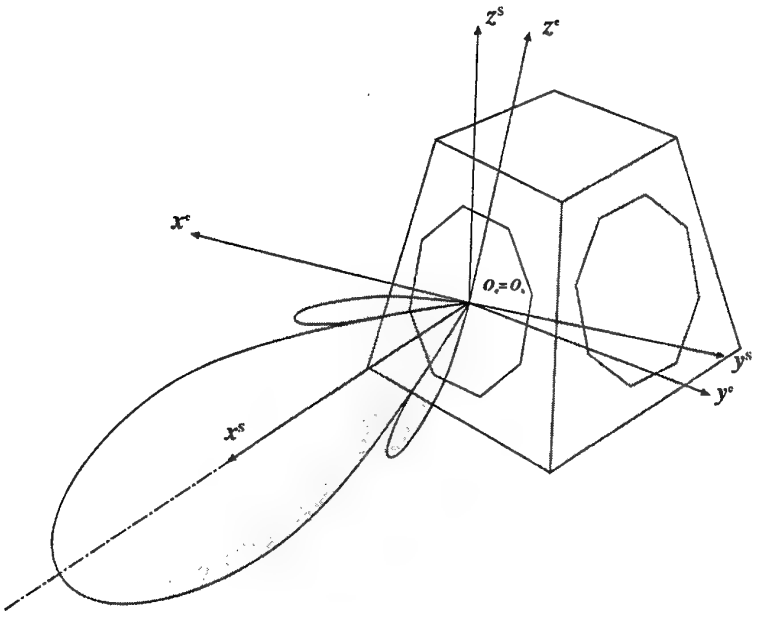


Figure 9.2b: The beam pattern of a phased array radar is illustrated. The coordinate system S is considered attached to the beam pattern. Its orientation in time is given by the BeamPatternOrientation and BeamPatternOrientationRate attributes of the SensorTask object instance. The coordinate system e is that attached to the component entity that is the phased array radar. See [1] for further details.

10. Representing the environment

Many of the federates that make up the Virtual Ship require environmental data. Sensor federates require data in order to model propagation. The motion federate, in its most general form, requires data concerning the surface wave field and local wind conditions.

The approach adopted to representing the environment may lie on a spectrum between two extremes. At one extreme all environmental information may be obtained via the RTI, and provided by dedicated environment federates. At the other extreme, no environment information is provided via the RTI and a federate accesses environment information locally.

The approach adopted in the VSA is that each federate that requires environment information accesses it locally, in a form optimum for that federate. No environment information is provided via the RTI.

This approach has been adopted for a number of reasons. In the first instance, different models often have different requirements for representing the environment. These may reflect different propagation algorithms used, or different levels of fidelity in representation of the environment. The lack of universally recognised representations of the environment imposes difficulty in constructing environment object classes within the VS-FOM to support exchange of environment information via the RTI.

An additional factor is recognition that some federates may make use of extremely large environmental data sets, such as meteorological data that may also vary in time. It would be inefficient to attempt to provide such large data sets via the RTI.

The approach adopted in the VSA offers considerable flexibility in representing the environment. A federate may utilise data of the required fidelity and in a format that is most convenient for its calculations. However, the consistency of environmental representation across the federation must be assessed during federation design.

11. The Name attribute of the CompositeEntity class

The Name attribute of the CompositeEntity class is a string intended to uniquely identify composite entities. In principle, the set of possible Name attributes will be extremely large. This set will also need to grow as the requirement emerges to represent new entities. The corollary of this is that the set needs to be constructed in such a way that additions can be easily made. Also of relevance is the observation that models of different fidelity require differing levels of knowledge concerning the identity of an entity. For example, a low fidelity federate representing an aircraft may simply require a description of an entity as a fighter, whereas a higher fidelity federate may require a description of the same entity as an F-18.

The solution to these requirements is to define the Name attribute as an element of a tree structure. This approach has a number of advantages. The use of a tree structure allows entities to be identified at various levels of fidelity. The tree structure can be easily added to. In addition, the tree structure can be easily navigated, an essential characteristic in scenario generation tools, for example, where an identity needs to be assigned to an entity.

The Name attribute is constructed as the concatenation of elements separated by ".". This is as follows:

Element_1.Element_2.Element_3.Element_4.Element_5.Element_6.Element_7.Element_8

Each element has a well defined meaning. These are as follows:

Element	Descriptor	Description
Element_1	Physical domain	Describes the physical domain in which the entity typically operates. This element echoes the names of the subclasses of the CompositeEntity object class.
Element_2	Military/Nonmilitary	Describes whether the entity is typically considered to be used in the military or non-military domains.
Element_3	Category	A broad grouping of entities that share common features.
Element_4	Template	A pattern for a collection of entities that is more specialised than the category.
Element_5	Country	The country that operates the entity.
Element_6	Type	A particular type of entity. This provides a description of an entity that may be useful for modelling purposes.
Element_7	Variant	The variant of a given type.
Element_8	Unique identifier	A unique and, generally speaking, common identifier for an entity. In the case of aircraft this will be a tail number. In the case of a warship, for example, this will be the common name, such as HMAS Adelaide.

The definitions of each term are enhanced by example. Consider the Name attributes of the RAN frigates HMAS Adelaide and HMAS Arunta respectively:

SeaSurface.Military.Warship.Frigate.Australia.FFG.Adelaide Class.HMAS Adelaide
SeaSurface.Military.Warship.Frigate.Australia.FFH.Anzac Class.HMAS Arunta

The set of possible Name attribute values forms a data set that supplements the VS-FOM. Indeed, given that all federates must be capable of dealing with the Name attribute of the composite entities that it discovers, this data set must be considered an essential adjunct to the VS-FOM. It thus assumes the same status as a standard data set and must be evolved and distributed in a controlled manner (see section 15). The evolution of the set of possible Name attributes is set in train through definition of a skeleton structure that guides further population of the set.

There are currently three branches of the tree, identified by the physical domain in which the entity exists. Current values of Element_1 in the Name attribute are:

Air, SeaSurface, SubSurface.

This set may be expanded as required.

Each of the three branches will now be expanded in turn, down to level 4. It should be borne in mind that at each level the set of possible values can be extended as required.

11.1 The Air branch

The first four levels of the Air branch of the tree structure is illustrated in Figure 11.1a.

11.1.1 Element_2 - Military/NonMilitary

Air.Military
Air.NonMilitary

11.1.2 Element_3 - Category

Air.Military.Countermeasure
Air.Military.FixedWing
Air.Military.RotaryWing
Air.Military.UAV
Air.Military.Weapon

11.1.3 Element_4 - Template

Air.Military.Countermeasure.Decoy

Air.Military.FixedWing.Bomber
Air.Military.FixedWing.Combat
Air.Military.FixedWing.Fighter
Air.Military.FixedWing.Maritime

Air.Military.FixedWing.Trainer
Air.Military.FixedWing.Transport
Air.Military.FixedWing.Utility

Air.Military.RotaryWing.Combat
Air.Military.RotaryWing.Shipborne
Air.Military.RotaryWing.Transport
Air.Military.RotaryWing.Utility

Air.Military.UAV.Tactical
Air.Military.UAV.Endurance

Air.Military.Weapon.Command Guided
Air.Military.Weapon.GPS Guided
Air.Military.Weapon.IR Guided
Air.Military.Weapon.Multi-Mode Guided
Air.Military.Weapon.RF Guided
Air.Military.Weapon.Unguided

11.2 The SeaSurface branch

The first four levels of the SeaSurface branch of the tree structure is illustrated in Figure 11.2a.

11.2.1 Element_2 - Military/NonMilitary

SeaSurface.Military
SeaSurface.NonMilitary

11.2.2 Element_3 - Category

SeaSurface.Military.Countermeasure
SeaSurface.Military.SupportShip
SeaSurface.Military.Warship

11.2.3 Element_4 - Template

SeaSurface.Military.Countermeasure.Decoy

SeaSurface.Military.SupportShip.Auxiliary
SeaSurface.Military.SupportShip.Service

SeaSurface.Military.Warship.Aircraft Carrier
SeaSurface.Military.Warship.Cruiser
SeaSurface.Military.Warship.Destroyer
SeaSurface.Military.Warship.Frigate
SeaSurface.Military.Warship.Landing
SeaSurface.Military.Warship.Mine Warfare
SeaSurface.Military.Warship.Patrol

11.3 The SubSurface branch

The first four levels of the SubSurface branch of the tree structure is illustrated in Figure 11.3a.

11.3.1 Element_2 - Military/NonMilitary

SubSurface.Military
SubSurface.NonMilitary

11.3.2 Element_3 - Category

SubSurface.Military.Countermeasure
SubSurface.Military.Sensor System
SubSurface.Military.Submarine
SubSurface.Military.Weapon
SubSurface.Military.UUV

11.3.3 Element_4 - Template

SubSurface.Military.Countermeasure.Decoy
SubSurface.Military.Countermeasure.Jammer

SubSurface.Military.Sensor System.Fixed Sonar Array
SubSurface.Military.Sensor System.Sonobuoy

SubSurface.Military.Submarine.Attack
SubSurface.Military.Submarine.Auxiliary
SubSurface.Military.Submarine.Strategic

SubSurface.Military.Weapon.Depth Charge
SubSurface.Military.Weapon.Lightweight Torpedo
SubSurface.Military.Weapon.Heavyweight Torpedo
SubSurface.Military.Weapon.Mine

SubSurface.Military.UUV.Mine Disposal Vehicle

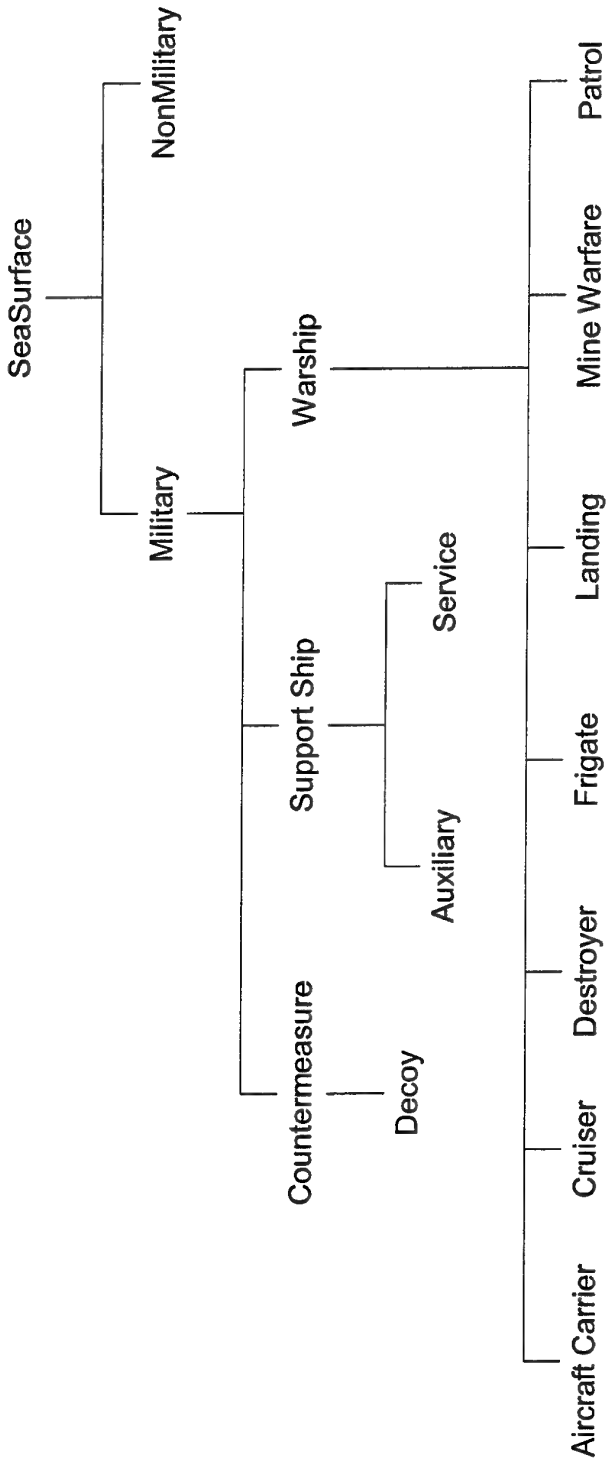


Figure 11.2a: The SeaSurface branch of the composite entity Name attribute data set. The first 4 levels of the tree are shown.

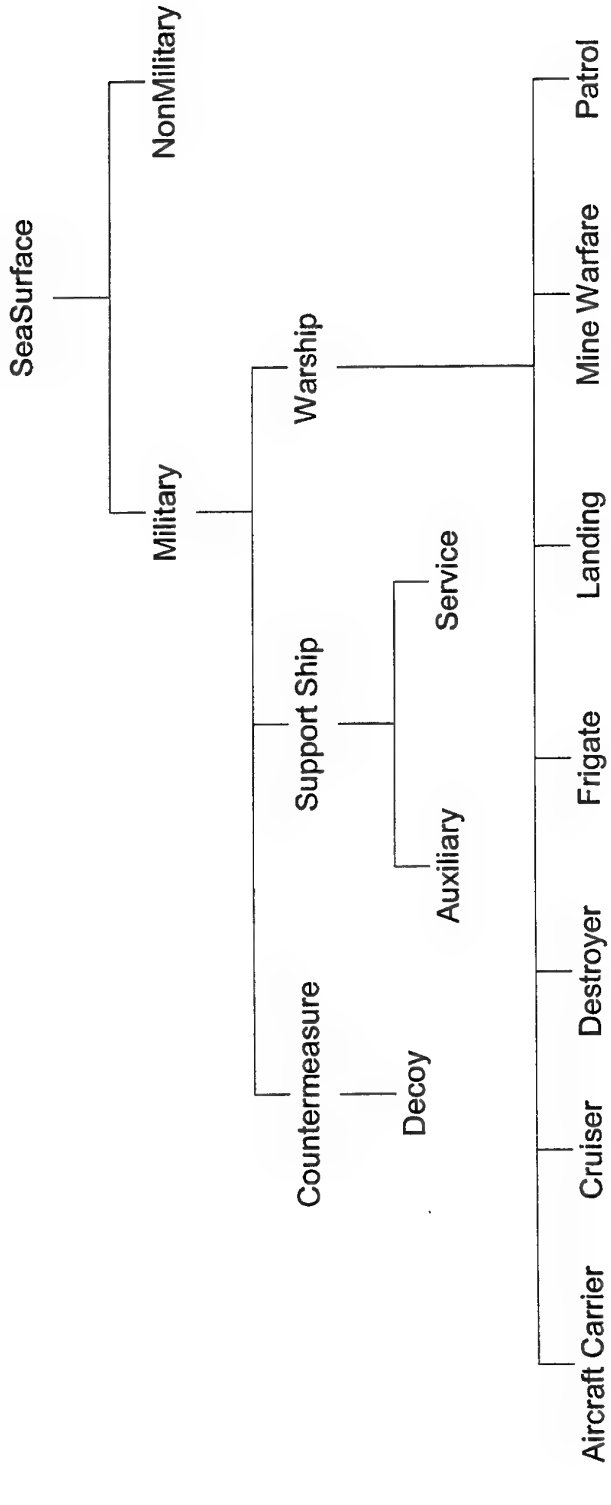


Figure 11.2a: The SeaSurface branch of the composite entity Name attribute data set. The first 4 levels of the tree are shown.

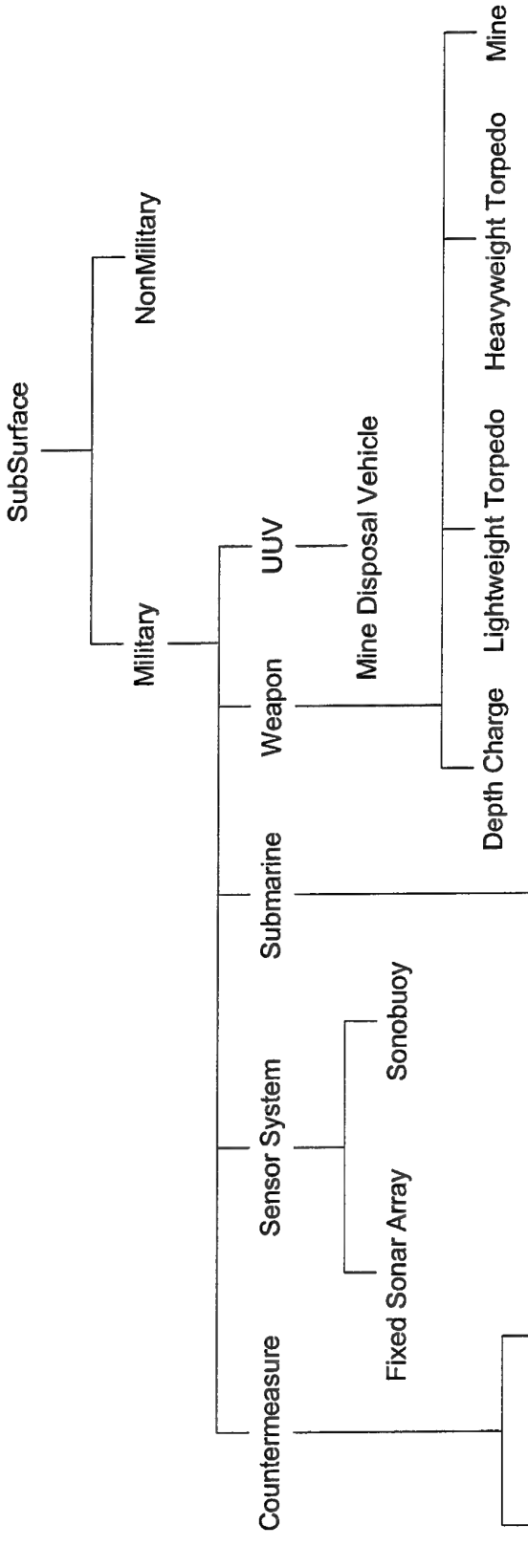


Figure 11.3a: The SubSurface branch of the composite entity Name attribute data set. The first 4 levels of the tree are shown.

12. The ComponentName attribute of the ComponentEntity class

The ComponentName attribute of the ComponentEntity class is a string intended to uniquely identify component entities. For reasons similar to those presented in the previous section, a tree structure is used to represent the set of all possible ComponentName attribute values.

The ComponentName attribute is constructed as the concatenation of elements separated by ".". This is as follows:

Element_1.Element_2.Element_3.Element_4.Element_5

Each element has a well defined meaning. These are as follows:

Element	Descriptor	Description
Element_1	System type	Describes the system type. This element echoes the names of the subclasses of the ComponentEntity object class.
Element_2	Category	A broad grouping of systems that share common features. If there is a class 3 element in the VS-FOM, this element echoes it.
Element_3	Template	A pattern for a collection of systems that is more specialised than the category. The template may reflect some common mode of operation of the system.
Element_4	Model	The particular model of a system.
Element_5	Variant	The variant of a given model.

The definitions of each term are enhanced by example. Consider the ComponentName attribute of a typical radar system:

SensorSystem.RadarSystem.PAR.Spy 1.D

As for the set of all possible Name attribute values, the set of possible ComponentName attribute values forms a data set that is an inseparable adjunct to the VS-FOM.

13. The Virtual Ship rules

In order to ensure interoperability of the federates that make up the Virtual Ship it is necessary to impose mandatory design characteristics. These are the Virtual Ship rules and are described in this section. The rules are classified as general rules or rules associated with implementation of the Virtual Ship execution management concept. The rationale for each of the rules is also presented.

13.1 General rules

13.1.1 General Rule 1 - Providing attribute updates upon request

All federates shall respond to the provideAttributeValueUpdate callback.

The response shall be as follows:

A federate shall immediately send a Receive Ordered (RO) updateAttributeValues message supplying those attributes requested. If any of the attributes being updated are time dependant then the tag argument of the updateAttributeValues message shall contain an encoded version of the time at which the time dependent attributes have been evaluated. If the federate is time regulating and has supplied time dependant variables Receive Ordered, it will additionally send a Time Stamp Ordered (TSO) updateAttributeValues message supplying those attributes as soon as possible.

13.1.1.1 Rationale for General Rule 1

The rationale for this rule can be best explained using an example. Suppose a federate F1 publishes objects and federate F2 subscribes to these objects. The objects published contain attributes that are time dependant. Consider the following sequence:

1. F1 receives a time advance grant to T1. F1 updates any registered objects to time T2 and requests a time advance to T2, thereby signifying that it will not provide any further messages with timestamp < T2 + Lookahead.
2. While waiting for the time advance to T2, F1 receives the federate ambassador callback provideAttributeValueUpdate. This is received in response to F2 calling the RTI service requestObjectInstanceAttributeUpdate.
3. The earliest possible time stamp F1 could associate with a Time Stamp Ordered updateAttributeValues message is T3 (>T2). However, the values of the attributes of the objects being modelled by F1 are only valid for time T2. Therefore, the only way F1 can respond to the provideAttributeValueUpdate callback immediately is to send a Receive Ordered updateAttributeValues message. As time dependant attributes are being sent, it is necessary to convey time information with the message. This is performed by encoding the time T2 in the tag argument of the updateAttributeValues RTI service (see section 13.1.2).
4. There is no constraint on when F2 will receive the reflectAttributeValues callback corresponding to the RO updateAttributeValues sent by F1. There is a chance that the message will be received in F2's future, ie, the time encoded in the tag argument is greater than F2's current time. In this case, any time dependant attributes are of limited use when they are received.

5. To account for the possibility that the RO updateAttributeValues message sent by F1 is received by F2 in its future, F1 will send an update of any time dependant variables in a TSO updateAttributeValues as soon as possible.
6. F1 receives a time advance grant to T2. F1 will then model its published objects to time T3 and provide a TSO updateAttributeValues with timestamp T3. F1 will then request a time advance to T3.
7. The TSO message sent by F1 will be received by F2 in accordance with the HLA time management algorithms. This will occur when F2 requests a time advance to a time T > T3.

From this example it can be seen that the need for sending an RO message immediately is due to the possibility of a sending federate having a very large timestep and hence not providing a TSO update for a considerable amount of time. A receiving federate can either wait for the TSO update to be received to obtain any time dependant attributes or can retreive them from the RO message and possibly store them until the receiving federate's time exceeds the time provided in the tag.

13.1.2 General Rule 2 - Encoding the time as the tag

When it is necessary to send time dependant variables in a Receive Ordered message, the time for which the variables are valid is to be encoded into the tag of the message as a human readable string. For example, if the time = 23.3, then the tag = "23.3".

13.1.2.1 Rationale for General Rule 2

A standard for encoding is required so that receivers may retrieve the time information. This rule provides that standard.

13.1.3 General Rule 3 - Providing kinematic attribute updates

When providing updates of kinematic attributes, which may be used in dead-reckoning algorithms, only the following possible combinations shall be provided:

CompositeEntity

Position
Position, Velocity
Position, Velocity, Acceleration
Orientation
Orientation, OrientationRate
Position, Orientation
Position, Orientation, OrientatationRate
Position, Velocity, Orientation
Position, Velocity, Orientation, OrientatationRate
Position, Velocity, Acceleration, Orientation
Position, Velocity, Acceleration, Orientation, OrientatationRate

SensorTask

BeamPatternOrientation
BeamPatternOrientation, BeamPatternOrientationRate

The principle followed is that for both linear and rotational motion, a single attribute update shall include all attributes up to the highest order time derivative that needs to be provided. A single attribute update shall be used to provide the required set of attributes.

13.1.3.1 Rationale for General Rule 3

This rule requires that if an n^{th} order time derivative of a kinematic attribute is being updated, the 0^{th}, 1^{st}, 2^{nd},..., $n\text{-}1^{th}$ order time derivatives are also provided in the same update. To understand the requirement for such a convention consider the case where the velocity only is provided. There is a potential lack of determinism about whether subsequent application of the dead reckoning formulae should use the dead-reckoned position at the time this update is received, or the value of the position when it was last updated. This rule removes any ambiguity and permits dead-reckoning using the most recently updated attribute values.

13.1.4 General Rule 4 - Use of big-endian

All data shall be sent to the RTI in big endian format.

13.1.4.1 Rationale for General Rule 4

Different platforms store data in memory in different byte ordering configurations. A standard must be adopted for communication between these platforms. The standard adopted by the Virtual Ship is to send data in big-endian format (also known as network byte ordering).

13.1.5 General Rule 5 - Sending arrays and complex datatypes

Sending multi-element data shall be accomplished by concatenating the byte representations of the individual data together. For complex datatypes the ordering of data in the sent byte array shall match the ordering of the complex datatype fields as they appear in the VS-FOM. The C++ string terminator character '\0' shall be used to terminate string data.

13.1.5.1 Rationale for General Rule 5

A receiving federate needs to know how to interpret the data it receives. The data received is in the form of an array of bytes with known length. A standard for how to fill this array with data must be adopted. This rule provides that standard.

In order that this data is decoded correctly, there needs to be understanding of the types of each of the data items sent (documented in the FOM) and also the size of each of the data types.

The following figures illustrate the concatenation of byte representations of data to form a multi element attribute update. In the figures, each square of storage represents a single byte.

Figure 13.1.5.1a illustrates sending three double values as a single attribute. Note that an array of three doubles in C++ or Java, or a C++ struct containing three doubles as fields will automatically have the required concatenated structure.

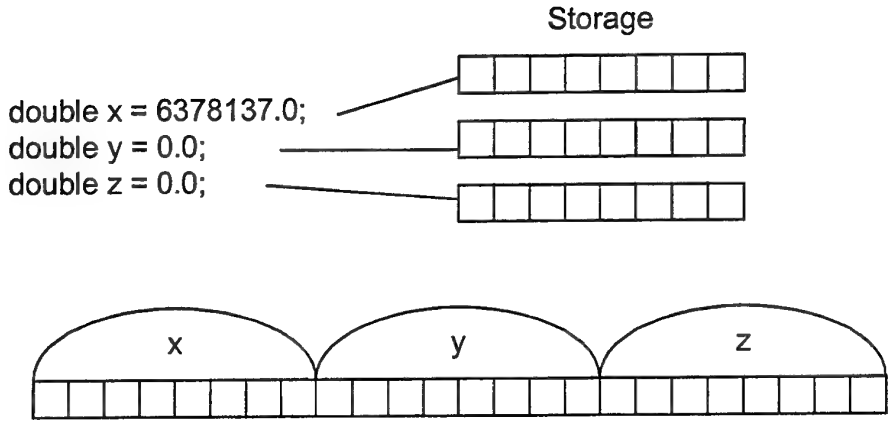


Figure 13.1.5.1a: Concatenating three doubles

Figure 13.1.5.1b illustrates sending a string and integer pair. The C++ string terminator '\0' is used to determine the end of a string.

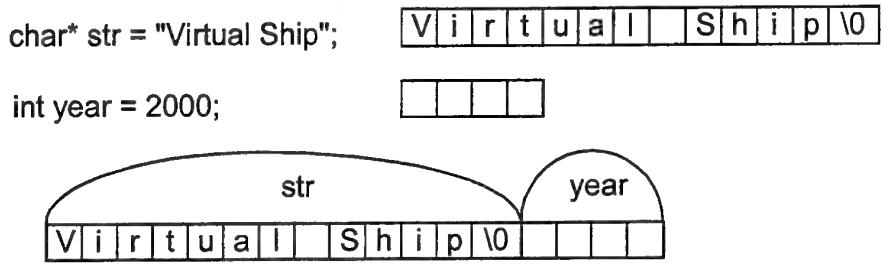


Figure 13.1.5.1b: Concetenating a string and an integer

Figure 13.1.5.1c illustrates sending an array of two string-integer pairs.

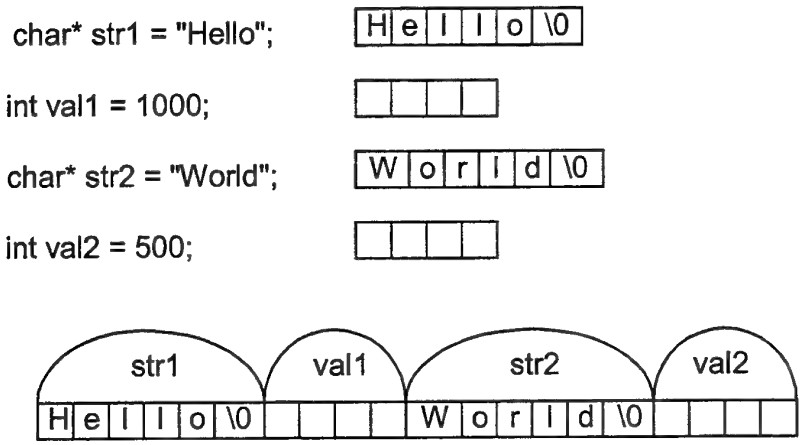


Figure 13.1.5.1c: Concatenating an array of two string-integer pairs

13.1.6 General Rule 6 - Boolean attributes

Boolean data shall be represented as a 4-byte integer with zero representing false and non-zero representing true.

13.1.6.1 Rationale for General Rule 6
A boolean value can be stored in a number of different ways. Representation as an integer is commonly used in C/C++.

13.2 Rules for an Execution Managed Virtual Ship Federation Execution

13.2.1 Execution Management Rule 1 - Handling initial attributes supplied in the ExecutionManagement.CreateCompositeEntity and ExecutionManagement.CreateComponentEntity interactions

An execution managed federate capable of creating an entity shall be able to process the initial attribute data provided by the ExecutionManagement.CreateCompositeEntity or ExecutionManagement.CreateComponentEntity interaction for all attributes in the VS-FOM that correspond to the object class of the entity created by the federate.

13.2.1.1 Rationale for Execution Management Rule 1
Problems can occur when the attributes provided in these interactions contain attributes that are not expected by the receiving federate. Specifying that all federates will be able to handle the attributes specified in the VS-FOM takes a step towards minimising this problem.

13.2.2 Execution Management Rule 2 - Clean-up at the end of a simulation loop

All object instances registered by a federate during a simulation must be deleted from the RTI at the end of the loop.

13.2.2.1 Rationale for Execution Management Rule 2
This rule allows for federates to remove their reflected images of registered object instances at the end of a simulation loop without having to wait for the removeObjectInstance callback. This may not be received until the start of the next simulation loop (if deleteObjectInstance was sent RO), if at all (if deleteObjectInstance was sent TSO the timestamp would have to be greater than the end time of the simulation which none of the federates exceed). It might be supposed that the rule could call for use the RO version of deleteObjectInstance at the end of a loop, but this does not ensure that remote federates will be notified of the removal before the start of the subsequent loop. It also must be borne in mind that federation restoration may remove all discovered instances from the LRC (since the save occurred when no instances had been discovered).

14. Time management

Within the Virtual Ship Architecture the unit of time is the second.

It is intended that federates within the Virtual Ship Architecture implement those time management policies that enable their correct operation. This view is in accordance with that of the HLA.

It must be borne in mind, however, that not all modes of time management are necessarily compatible. For example, a federate that does not use time management services and advances based on local wall clock time, may have a time that bears no correlation with that of federates that are time constrained and time regulating. The consistency of time management strategies across a federation must be determined during federation design.

An issue that arises in terms of time management concerns execution management. In the execution management concept, the VSEM sends interactions that instruct federates to create instances of the CompositeEntity and ComponentEntity object classes. The VSEM is time regulating and time constrained and these interactions are sent in time stamp order (TSO). For a managed federation execution to be repeatable all other federates must be time regulating and time constrained. This ensures that the federation cannot advance until these interactions have been received and the relevant object instances registered, and an initial attribute update provided.

In the event that some federates in a managed federation execution do not utilise time management services, strict repeatability is lost. However, the use of synchronisation points to control the flow of the federation execution, and particularly entry to the simulate phase of the federation execution (see Figure 8.1), provides for a common scenario start time, within limits that reflect network latency.

15. Evolution of the Virtual Ship Architecture

The Virtual Ship Architecture in its current form is not complete. It is intended that as it is exercised, and as new applications come to light , it will be evolved to reflect users experience and support new applications.

A simplified view of the process that will drive further evolution of the Virtual Ship Architecture is illustrated in Figure 15a. New applications of the Virtual Ship are identified and a design that satisfies the simulation requirement is proposed, in accordance with the broad concept of what the Virtual Ship is, as defined in this document. Extensions to the VSA required to implement this instance of Virtual Ship are proposed, and an implementation phase follows. During implementation practical necessity drives small scale modification of the proposed components of the architecture. When the instance of Virtual Ship is completed, the lessons learnt have been incorporated into that iteration of the architecture and another cycle commences, beginning with identification of a new application.

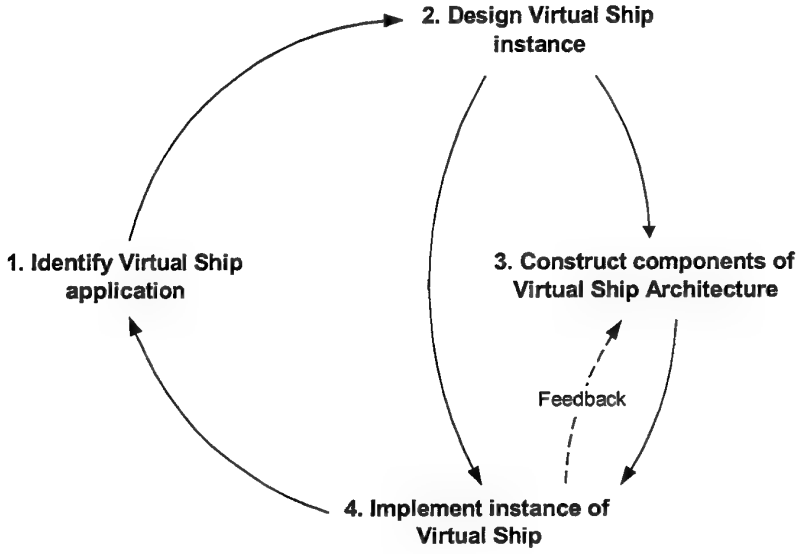


Figure 15a: The interaction between Virtual Ship Architecture development activity and creation of instances of the Virtual Ship.

In essence, at any particular time there will be an authorised version of the VSA and experimental versions that reflect the requirements of implementing new applications. At particular times proposals will be made for changes to the authorised version of the VSA. Such proposals shall be considered in light of the experience derived from those activities that require additions to, or amendments of, the authorised version of the VSA.

The forum for approving such changes is the Virtual Ship Architecture Working Group (VSAWG). The VSAWG is a joint DSTO-Industry forum formed explicitly for this purpose. The operation of the VSAWG is governed by Terms of Reference attached as Annex B.

15.1 Strategies for evolution of the VS-FOM

There are a number of characteristics of the FOM that will facilitate its evolution in a controlled manner and that, to a great extent, provide backward compatibility.

In the first instance, branches of the object class structure can essentially be considered in isolation. As an example, consider the existing VS-FOM. The CompositeEntity branch facilitates exchange of information concerning composite entities, and this requirement may be considered separately from that for exchanging data concerning component entities. The only link between the two is via the ParentCompositeEntityObjectInstanceName attribute of the ComponentEntity class that permits association of component entities with their parent entity.

As a consequence of this, blocks of the FOM may be developed in isolation and later merged. Indeed, if we consider the current VS-FOM, development of the Track branch may proceed independently of others. If new applications are identified that require data exchange beyond that supported by the existing VS-FOM, then a strategy for fulfilling the requirement is to develop a new branch of the object class structure.

The hierarchical structure of the FOM also enables a strategy to VS-FOM evolution to be adopted in which the hierarchy is extended to enable exchange of more specialised data concerning entities. To illustrate by way of an example, consider the component entity branch of the object class structure table.

ComponentEntity (N)	CommandAndControlSystem (PS)		
	CountermeasureSystem (PS)		
	NavigationSystem (PS)		
	SensorSystem (N)	ESMSystem (PS)	
		IRSystem (PS)	
		IFFSystem (PS)	
		RadarSystem (PS)	SpecialisedRadarSystem (PS)
		SonarSystem (PS)	
	WeaponSystem (PS)		

If it were necessary to exchange additional data to describe a specialised radar system, say, then a subclass of the RadarSystem class could be introduced. This is shown as the SpecialisedRadarSystem in the above section of the VS-FOM. Through the inheritance property, federates that subscribe to the ComponentEntity.SensorSystem.RadarSystem object class will discover the ComponentEntity.SensorSystem.RadarSystem.SpecialisedRadarSystem object instance as a ComponentEntity.SensorSystem.RadarSystem object instance and continue to model its interaction with the radar, albeit at possibly a lower level of fidelity.

15.2 Issues to be addressed in development of the VSA

Through development of this version of the VSA a number of issues have been identified that will be addressed in future iterations of the VSA. Key issues are outlined here.

15.2.1 Use of Save and Restore within the execution management concept

As noted in the document *Execution Management in the Virtual Ship Architecture* [2], the use of federation save and restore functions as key components of the execution management concept may restrict the use of third party federates in Virtual Ship federation executions.

Methodologies for surmounting this difficulty need to be explored. A partial solution may be for a managed federation to only be saved in the event that multiple replications of a scenario are required within the federation execution. If a single execution of the scenario is to take place, such as when a human-in-the-loop scenario is to occur, then the federation need not be saved. Such an approach would permit use of federates that do not implement the save and restore functions.

15.2.2 Identification of system state data

At present there are no attributes of the subclasses of the ComponentEntity class. The role that this class structure currently plays is one of providing for selective discovery of specific types of component entity. The intention, though, is that the attributes of the subclasses shall convey the state of the particular systems. Future applications will drive the process of identifying these.

15.2.3 Identification of system control messages

As the Virtual Ship expands to consider scenarios in which there is significant command and control exercised over the systems that compose a warship, it will be required to support the exchange of system control messages. These will be interactions and the class structure shall be used to group messages that have some common characteristics. As for system state data, future applications will drive the process of identifying these.

15.2.4 Representation of the environment

The current version of the VSA calls for individual federates to access environment data locally, in the form and fidelity required by them. It is anticipated that as the Virtual Ship is used common ways of representing the environment will emerge. If this occurs, it may be appropriate to support the exchange of environment data via the RTI.

15.2.5 Modelling propagation

The current version of the VSA calls for federates representing sensors to model the propagation of signals relevant to determining the sensor output. It has been recognised that significant advantages may be conferred if propagation can be modelled within dedicated federates. However, the means of achieving such a circumstance are far from clear and some of the issues have been identified in [5] and [6]. It is anticipated, though, that the future will see some resolution of these issues and the emergence of commonly recognised ways of representing signal propagation in a way that is consistent with the notion of a central propagation federate. When such a circumstance has been achieved the VSA will be modified to accommodate it.

15.2.6 Mapping between the VS-FOM and other FOMs

A critical issue in the HLA community is that of FOM-agility, or the ability to map between different FOMs. An initial challenge will be to map between the VS-FOM and the RPR-FOM that encapsulates the data exchanged through the DIS protocol. The RPR-FOM is in common use and the ability to map between the two is expected to provide increased utility to Virtual Ship federations through the ability to make use of RPR-FOM based federates. Similarly, Virtual Ship federates may be used within RPR-FOM based federations.

15.2.7 Mapping between the VS-FOM and DIS protocol data units

Although the US has imposed a HLA mandate, applications remain that use the DIS protocol to achieve simulation interoperability. In the interests of extending the applicability of Virtual Ship federates it is appropriate to explore the mapping between the VS-FOM and DIS protocol data units. The solution to this problem is essentially the same as for the RPR-FOM, given that the RPR-FOM has been constructed to encapsulate the data exchanged through the DIS protocol.

16. References

1. Best, J.P. *Coordinate Usage in the Virtual Ship Architecture. Issue 1.00.* DSTO-GD-0260 (2000).
2. Cramp, A. & Best, J.P. *Execution Management in the Virtual Ship Architecture. Issue 1.00.* DSTO-GD-0258 (2000).
3. http://hla.dmso.mil
4. http://www.dmso.mil/documents/policy/nato_msmp/index.html
5. Best, J. P. & Luckman, N. *A Concept for a HLA Compliant Propagation Federate. Part 1. Using the Object Class Structure.* In: Proceedings of SimTecT 2000 (2000).
6. Cramp, A. *A Concept for a HLA Compliant Propagation Federate. Part 2. Using Interactions to Represent Signal Propagation.* In: Proceedings of SimTecT 2000 (2000).

17. Acknowledgements

Development of the Virtual Ship Architecture has benefited significantly from the input of the Virtual Ship Architecture Working Group (VSAWG). Notable contributions have been made by Rob Zeltzer, Kevin Gaylor and Marcel Scholz.

18. List of acronyms

Acronym		First reference (Section)	Explanatory reference (Section)
ALSP	Aggregate Level Simulation Protocol	3	N/A
API	Application Programmers Interface	3	N/A
DIS	Distributed Interactive Simulation Protocol	3	N/A
DMSO	Defense Modeling and Simulation Office	3	N/A
DSTO	Defence Science and Technology Organisation	15	N/A
ESM	Electronic Support Measures	5.4.1	5.4.1
FOM	Federation Object Model	3	3
HLA	High Level Architecture	1	3
IFF	Interrogate Friend or Foe	6.1.2	N/A
IR	Infrared	6.1.4	N/A
IRST	Infrared Search and Track	7.2	N/A
LRC	Local RTI Component	3	3
NATO	North Atlantic Treaty Organisation	3	N/A
OMT	Object Model Template	3	3
RCS	Radar Cross Section	6.1.4	N/A
RF	Radio Frequency	5.2.2	N/A
RO	Receive Order	13.1.1	N/A
RPR-FOM	Realtime Platform Reference FOM	15.2.6	N/A
RTI	Runtime Infrastructure	3	3
SNR	Signal-to-Noise Ratio	6.1.4	N/A
SOM	Simulation Object Model	3	3
TSO	Time Stamp Order	8	N/A
UAV	Unmanned Aerial Vehicle	11.1	N/A
UUV	Unmanned Underwater Vehicle	11.3	N/A
VSA	Virtual Ship Architecture	1	4
VSAWG	Virtual Ship Architecture Working Group	15	Annex B
VSEM	Virtual Ship Execution Manager	7.2	7.2
VS-FOM	Virtual Ship Federation Object Model	2	6
VSSD	Virtual Ship Simulation Display	7.2	7.2
WGS84	World Geodetic System 1984	6.1.1	N/A

Annex A: The requirement for the Virtual Ship Architecture, as at 21 May 1999

Introduction

The Virtual Ship concept is based upon the integration of simulation models that represent the components of a warship. To facilitate this, the High Level Architecture (HLA) is being utilised. The HLA enables models to be linked over a computer network. The particular processes that the HLA supports are the exchange of data amongst the models and the coordination of their time advances. Within the HLA a simulation model is known as a federate and a collection of federates operating together is a federation.

The HLA calls for the existence of an entity known as the Run Time Infrastructure (RTI). In an abstract sense, the RTI provides the mechanism for exchanging data amongst federates and coordinating their time advances. In practice, the RTI is implemented as software.

In a formal sense the HLA has three components. These are

1. The Rules,
2. The Object Model Template (OMT),
3. The Interface Specification.

The rules describe mandatory characteristics of the federates and federations. The Object Model Template is a format for describing the data that is either made available and required by an individual federate, or all the data that is exchanged over the network by all federates participating in a federation execution. The description of the data provided and required by a federate is known as the Simulation Object Model (SOM). The description of all the data exchanged amongst all federates is known as the Federation Object Model (FOM). Both the SOM and FOM are constructed in accordance with the OMT. The Interface Specification describes in formal terms how federates interact with the RTI.

The Virtual Ship Architecture (VSA)

The Virtual Ship is a specific application exploiting the HLA. A conceptual representation of it is illustrated in Figure 1. The typical federates that compose the Virtual Ship are shown here and include:

1. Sensor systems,
2. Weapon systems,
3. Countermeasure systems,
4. Navigation systems,
5. C^2 systems,
6. Motion models,
7. Damage models,
8. External entity models,

9. Scenario control systems.

The models that constitute it will have certain common characteristics and will exchange specific data items. The generic classes of data exchanged include:

1. Entity type data,
2. Entity kinematic data,
3. System status data,
4. Tactical data,
5. System control messages,
6. Scenario control messages,
7. Weapons impact/detonation notification messages,
8. Damage indication messages.

In Figure 1, a solid line indicates the exchange of information describing persistent entities. Within the HLA this data exchange is effected through the mechanism of an object attribute update. The dashed lines in the figure indicate the exchange of information that has a transient quality. Within the HLA this data exchange is effected through the mechanism of an interaction.

The Virtual Ship Architecture (VSA) refers to those components that customise the HLA for use in linking simulation models as illustrated in Figure 1. The components of the VSA are envisaged to be as follows.

1. ***The Virtual Ship FOM (VS-FOM).*** The VS-FOM will define the data exchange standard for those federates that constitute the Virtual Ship. It will be constructed in accordance with the Object Model Template (OMT). The VS-FOM will exploit class hierarchies in order to support federates of differing fidelity. The VS-FOM will be constructed in such a manner that it readily enables the incorporation of additional object and interaction classes.
2. ***The Virtual Ship rules.*** The Virtual Ship rules will describe mandatory characteristics of federates brought into the Virtual Ship, over and above the HLA rules.
3. ***The Virtual Ship lexicon.*** The objects, attributes, interactions and parameters within the VS-FOM will form the Virtual Ship lexicon. The lexicon will provide for common terminology use across modelling applications. It is through the common interpretation of data that a common world view can be established amongst distributed federates.
4. ***The Virtual Ship data standards.*** Different modelling applications relevant to the Virtual Ship have shared interest in data. The provision and exploitation of this data will be facilitated through adoption and establishment of common data standards.
5. ***The Virtual Ship tools.*** Tools will be adopted and developed in order to: facilitate the construction of federations in support of specific applications, manage federation executions, gather and analyse data during federation executions, test federates and federations, facilitate sharing and common interpretation of data describing federates and federations.

Background

The Virtual Ship Architecture will evolve to support new modelling applications and increasing levels of fidelity. As a consequence, what will bound the development of successive versions will be the breadth of application and the degree of fidelity supported. Although broad guidelines may be stated for bounds that define a particular version, an exact specification of what constitutes a particular version cannot be given in advance.

The purpose in developing the VSA is to enable simulation of warship operations. It is required to capture the dynamic interactions between the systems that constitute a warship, the people who operate them and the external entities that constitute the threat environment.

Those that operate a warship obtain their appreciation of the current situation primarily through the outputs of their organic sensors. As a consequence, the VSA will adopt a model of the world that is centred on the inputs to, and outputs from, the variety of sensors that are utilised in maritime warfare. This view is referred to as a "sensor centric" view of the problem space.

Compliance with the VSA is a necessary condition for federates to operate as components of the Virtual Ship. The establishment of a sufficiency condition will remain the responsibility of those developing Virtual Ship federations in support of specific applications.

Requirement for Version 1.0 of the Virtual Ship Architecture (VSA 1.0)

Version 1.0 of the Virtual Ship Architecture will have the following characteristics.

1. VSA 1.0 will facilitate the linking of models of ship components at the system level. The system models that may be linked are:

Sensor systems, including
- Radar systems,
- Sonar systems,
- Electronic support systems,

Weapon systems,
Countermeasure systems,
Navigation systems,
Command and control system components, including
- Sensor data fusion,
- Threat evaluation,
- Weapon assignment,
- Fire control,
- Operator aids,
- Command decision aids.

Other models that may be linked include:

Ship motion,
Weapon damage.

2. VSA 1.0 will facilitate the linking of federates that

Represent external entities,
Generate and control scenarios,
Display the entities that exist within a scenario.

3. VSA 1.0 will facilitate the exchange of the following broad classes of data between federates:

Entity type data,
Entity kinematic data,
System status data,
Tactical data, including detections and tracks,
System control messages,
Scenario control messages,
Weapon impact/detonation notification messages,
Damage indication messages.

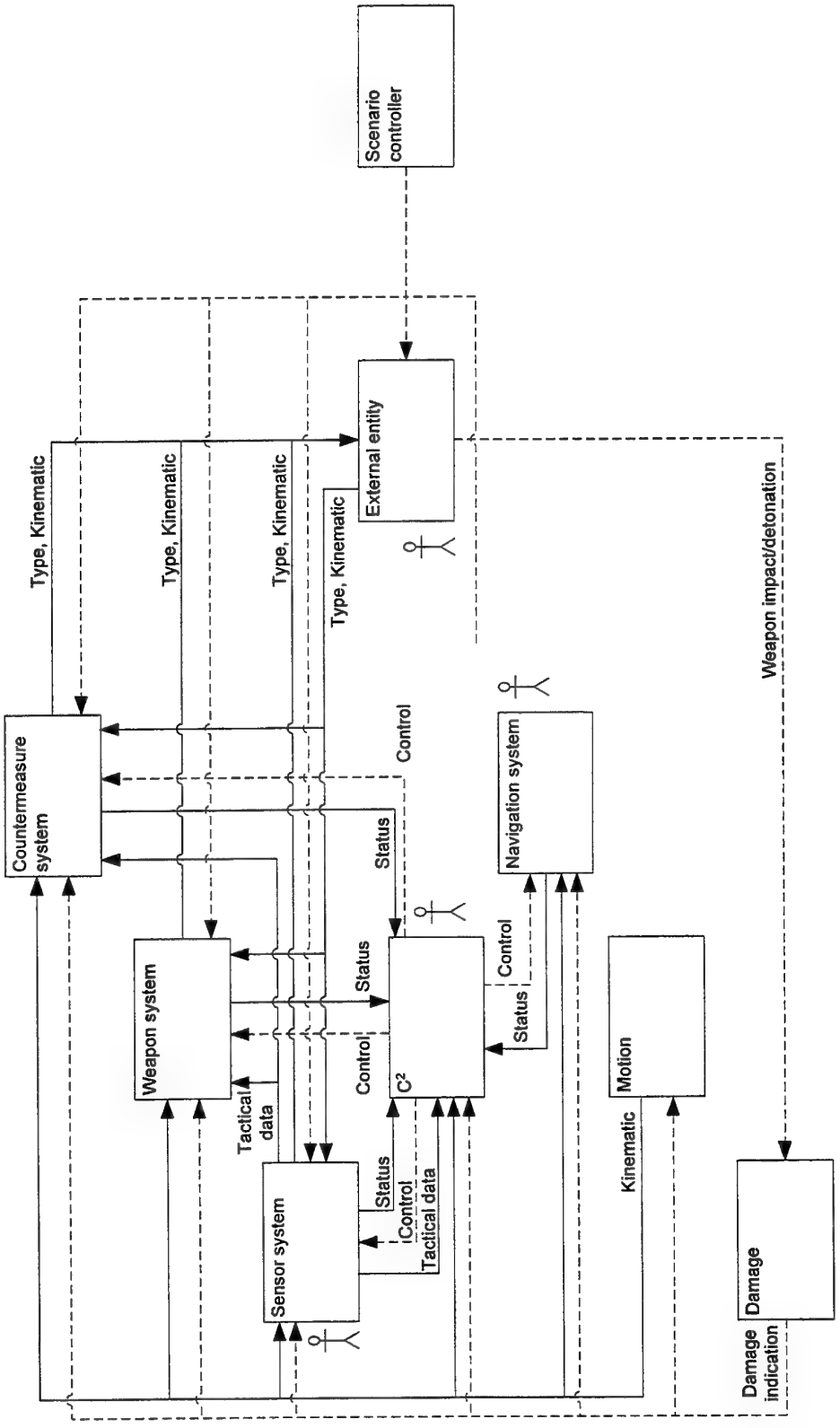
4. Within VSA 1.0, propagation of electromagnetic and acoustic signals shall be modelled within federates that model sensors. This requirement provides the principal bound on the fidelity with which systems and the interactions between them are modelled.

5. VSA 1.0 will facilitate interoperability with federates that model at the entity level and which achieve this through compliance with the RPR-FOM.

6. VSA 1.0 will utilise a class hierarchy that supports later enhancements in the fidelity with which systems and other entities are modelled. The class hierarchy will facilitate backwards compatibility as the VSA evolves.

7. VSA 1.0 will be developed with a view to later versions enabling the following types of simulation models to be linked:

Communication networks, including
  Intra-ship networks,
  Inter-ship networks,
  Tactical data links,
  Strategic data links,
Software agents,
Propagation,
Electronic attack,
Infrared sensors.


Scenario controller
External entity
Type, Kinematic
Type, Kinematic
Type, Kinematic
Type, Kinematic
Weapon impact/detonation
Countermeasure system
Control
Navigation system
Status
Weapon system
Status
Control
Status
Control
C²
Motion
Tactical data
Kinematic
Status
Control
Tactical data
Sensor system
Damage
Damage indication

Annex B: Virtual Ship Architecture Working Group (VSAWG) Terms of Reference, as at 21 May 1999

The role of the Virtual Ship Architecture Working Group (VSAWG) is to determine, document and evolve the architecture of the Virtual Ship. Having established a baseline version, the VSAWG will evolve the Virtual Ship architecture in order to maximise the utility of the Virtual Ship. The Virtual Ship architecture will be subject to configuration control.

The VSAWG will operate in a consultative manner and ensure that the technical content of the architecture is guided by the requirements of the user community. Ultimate responsibility for the content of the architecture will reside with the VSAWG.

The architecture of the Virtual Ship will be based upon the High Level Architecture (HLA)†. Models of system components will be the federates of a HLA federation. The federation will be described by the Virtual Ship FOM (VS-FOM), and the development of this will be the principal task of the VSAWG.

Elements of the Virtual Ship Architecture (VSA)

The Virtual Ship will be constructed in accordance with the High Level Architecture. In addition, the specific objective of simulating warship operations necessitates addition to, or specialisation of, the HLA. This collection of additions and specialisations will constitute the Virtual Ship Architecture (VSA).

There will be five essential elements of the VSA. These are described as follows.

The VS-FOM

This defines the data exchange contract amongst the federates that constitute the Virtual Ship. It will be constructed in accordance with the Object Model Template (OMT).

† The High Level Architecture is a framework for performing distributed simulation. A number of simulation models, known as federates, operate together to create a common simulated environment. The collection of all federates is known as the federation. In order to operate together the federates must exchange information. The data exchanged amongst federates is described by the Federation Object Model (FOM). The structure of the FOM is determined by the federation developer and federates must be engineered to exchange data in accordance with it. The format of the FOM is given by the Object Model Template (OMT). This is a tabular format that details the objects within the federation, the attributes that describe them, and the interactions amongst the objects which are described by parameters.

The Virtual Ship Lexicon

The objects, attributes, interactions and parameters within the VS-FOM will form the Virtual Ship lexicon. The lexicon will provide for common terminology use across modelling applications. It is through the common interpretation of data that a common world view can be established amongst distributed federates.

The Virtual Ship Rules

The Virtual Ship rules will describe mandatory characteristics of federates brought into the Virtual Ship, over and above the HLA rules.

Virtual Ship Data Standards

Different modelling applications relevant to the Virtual Ship have shared interest in data. The provision and exploitation of this data will be facilitated through establishment of common data standards.

The Virtual Ship Tools

A variety of existing tools will support development of the Virtual Ship. In addition to tools that support the HLA generally, a requirement for additional tools can be foreseen. These will support scenario generation and control, federate development, object model consistency checking, federate monitoring including stealth viewing and federation execution management and analysis.

Outputs

The VSAWG will produce the following:

1. A VS-FOM documented in accordance with the HLA OMT,
2. A report describing the principal elements of the VSA, with particular reference to the VS-FOM,
3. A document describing the manner in which the VS-FOM will evolve, including configuration control,
4. A series of working papers documenting the technical considerations on topics of relevance to whole ship simulation,
5. A technical brief on the VSA, which is public release,
6. A non-technical brief on the VSA, which is public release.

Key tasks

The essential element of the VSA is the VS-FOM. Construction of this will drive activity related to the other four aspects of the VSA. The task in constructing the VS-FOM is to determine the objects that will be represented in the Virtual Ship from a data exchange point of view, and the interactions amongst the federates. In the lexicon of the HLA the objects are described by their attributes and interactions are described by their parameters.

The tasks that must be performed by the VSAWG in order to capture these data are to:

1. Identify the ship components that will be modelled and the interactions amongst them,
2. Identify the requirement for data exchange with the outside world, or the scenario,
3. Determine a set of objects and interactions that capture this requirement,
4. Determine the attributes and parameters of the objects and interactions,
5. Determine a lexicon of objects, interactions, attributes and parameters,
6. Determine mandatory requirements (rules) for federates to participate in the Virtual Ship federation executions,
7. Define key enumerated data types and complex data types,
8. Construct the VS-FOM, exploiting object and interaction class hierarchies,
9. Assess the VS-FOM for robustness with respect to the introduction of new attributes, higher fidelity models, lower fidelity models, new object classes and new interaction classes,
10. Identify common data requirements across simulation applications,
11. Identify/specify software tools that will assist in the development of simulation models that are compliant with the VS-FOM.

Process

The VSAWG will meet on a regular basis, at an interval to be determined by the members. Additional meetings may be called as required. Meetings of the full VSAWG are intended primarily as a mechanism for providing rigorous consideration of architecture proposals and decision making concerning these.

The bulk of the technical work involved in formulating the Virtual Ship Architecture will occur out of session. A number of subcommittees will be formed in order to address the data exchange requirement amongst models of various ship systems. For example, a subcommittee may be formed to consider radar modelling and another to consider above water weapons. There will be cross membership of subcommittees to exploit system commonalities. For example, passive sonar and ESM share many similarities and their joint consideration offers the prospect of accelerated progress.

The outcome of subcommittee deliberations will be a series of working papers. These might be profitably published as DSTO Technical Notes, with due recognition of Industry contributions.

These findings shall be brought together by the VSAWG in order to determine the Virtual Ship Architecture. The full VSAWG will subject the architecture proposals of the subcommittees to rigorous analysis, particularly with respect to the issue of robustness as noted above.

DISTRIBUTION LIST

Virtual Ship Architecture Description Document - Issue 1.00

Issued by John P. Best

AUSTRALIA

DEFENCE ORGANISATION

S&T Program

Chief Defence Scientist
FAS Science Policy
AS Science Corporate Management
} shared copy

Director General Science Policy Development
Counsellor Defence Science, London (Doc Data Sheet)
Counsellor Defence Science, Washington (Doc Data Sheet)
Scientific Adviser to MRDC Thailand (Doc Data Sheet)
Scientific Adviser Policy and Command
Navy Scientific Adviser
Scientific Adviser - Army (Doc Data Sheet and distribution list only)
Air Force Scientific Adviser
Director Trials

Aeronautical and Maritime Research Laboratory
Director

Electronics and Surveillance Research Laboratory
Director

CMOD
RL-MODSA
Dr J. Best (MOD)
A. Cramp (MOD)
Dr J. Legg (SSD)
N. Luckman (WSD)
Dr S. Canney (Doc Data Sheet EWSTIS)
Dr D. Sutton (WSD)
B. Hermans (Doc Data Sheet EWSTIS)
G. Horsfall (Doc Data Sheet EWSTIS)
M. Saunders (WSD)
Dr A. Anvar (MOD)

DSTO Library and Archives

Library Fishermans Bend (Doc Data Sheet)
Library Maribyrnong (Doc Data Sheet)
Library Salisbury
Australian Archives
Library, MOD, Pyrmont
Library, MOD, HMAS Stirling
US Defense Technical Information Center, 2 copies

UK Defence Research Information Centre, 2 copies
Canada Defence Scientific Information Service, 1 copy
NZ Defence Information Centre, 1 copy
National Library of Australia, 1 copy

Capability Development Division
Director General Maritime Development
Director General C3I Development (Doc Data Sheet only)
Director General Aerospace Development (Doc Data Sheet only)

Navy
Capability Development Manager, SCFEG, Building 90, Garden Island
SO (Science), COMAUSNAVSURFGRP, Garden Island (Doc Data Sheet only)

Army
ABCA Office, G-1-34, Russell Offices, Canberra (4 copies)
SO (Science), DJFHQ(L), MILPO Enoggera, Queensland 4051
(Doc Data Sheet only)
NAPOC QWG Engineer NBCD c/- DENGRS-A, HQ Engineer Centre Liverpool
Military Area, NSW 2174 (Doc Data Sheet only)

Intelligence Program
DGSTA Defence Intelligence Organisation
Manager, Information Centre, Defence Intelligence Organisation

Corporate Support Program
Library Manager, DLS-Canberra (Doc Data Sheet)

UNIVERSITIES AND COLLEGES
Australian Defence Force Academy
Library
Head of Aerospace and Mechanical Engineering
Serials Section (M list), Deakin University Library, Geelong, 3217
Senior Librarian, Hargrave Library, Monash University
Librarian, Flinders University

OTHER ORGANISATIONS

NASA (Canberra)
AusInfo

OUTSIDE AUSTRALIA

ABSTRACTING AND INFORMATION ORGANISATIONS
Library, Chemical Abstracts Reference Service
Engineering Societies Library, US
Materials Information, Cambridge Scientific Abstracts, US
Documents Librarian, The Center for Research Libraries, US

INFORMATION EXCHANGE AGREEMENT PARTNERS

Acquisitions Unit, Science Reference and Information Service, UK
Library - Exchange Desk, National Institute of Standards and Technology, US

SPARES (5 copies)

Total number of copies: **54**

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE

Virtual Ship Architecture Description Document
Issue 1.00

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)

Document	(U)
Title	(U)
Abstract	(U)

4. AUTHOR(S)

Issued by
John P. Best

5. CORPORATE AUTHOR

Aeronautical and Maritime Research Laboratory
PO Box 4331
Melbourne Vic 3001 Australia

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-GD-0257	AR-011-612	General Document	October 2000

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
M9505-19-164	NAV 98/173	DGMD	126	6

13. URL on the World Wide Web

http://www.dsto.defence.gov.au/corporate/reports/ DSTO-GD-0257.pdf

14. RELEASE AUTHORITY

Chief, Maritime Operations Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, SALISBURY, SA 5108

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CASUAL ANNOUNCEMENT Yes

18. DEFTEST DESCRIPTORS

Command and control systems, Computerized simulation, Computer architecture, Naval warfare, Virtual reality

19. ABSTRACT

The Virtual Ship concept calls for simulation models of systems to be brought together to create a virtual representation of a warship, in a process analogous to the construction of a physical ship. The essential technical requirement enabling the Virtual Ship is an architecture for linking simulation models. The High Level Architecture (HLA) has been adopted and extended for this purpose. This extension is known as the Virtual Ship Architecture (VSA), and is described in this document.